Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

3D First-Person Logic Game

Bachelor thesis

Lucie Veverková

Field of study: Computer Games and Graphics
Supervisor: Ing. Ladislav Čmolík, Ph.D.

Prague, May 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Veverková** Jméno: **Lucie** Osobní číslo: **483797**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

3D logická hra z pohledu první osoby

Název bakalářské práce anglicky:

3D first-person logic game

Pokyny pro vypracování:

Seznamte se s principynávrhu počítačových her. Proveďte analýzu herních principů používaných v 3D logických počítačových hrách. Dále se seznamte modulárními komponentami a jejich využitím při návrhu úrovní. Na základě analýzy vytvořte design dokument pro 3D logickou počítačovou hru. Dále vytvořte modulární komponenty, ze kterých se budou skládat jednotlivé úrovně, jejich materiály a textury. Dle design dokumentu vytvořte s využitím modulárních komponent alespoň tři hratelné úrovně hry. Výslednou hru otestujte pomocí kvalitativních testů alespoň s šesti hráči.

Seznam doporučené literatury:

[1] R. Koster. Theory of Fun for Game Design, 2nd edition, O'Reilly Media, 2013.
[2] J. Schell. The Art of Game Design: A book of lenses. CRC Press, 2008.
[3] B. L. Mitchell. Game Design Essentials, John Wiley & Sons, 2012.
[4] S. Rogers. Level up! the Guide to Great Video Game Design, John Wiley & Sons, 2014.
[5] E. De Nucci and A. Kramarzewski. Practical Game Design : Learn the art of game design through applicable skills and cutting-edge insights, Packt Publishing, 2018.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ladislav Čmolík, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **18.03.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **19.02.2023**

Ing. Ladislav Čmolík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studentky

Declaration

I hereby declare I have written this Bachelor thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis.

In Prague, May 2021

.....
Lucie Veverková

Abstract

This thesis provides an analysis of the best principles of creating a good puzzle game, the overall process of level design, the most common modeling techniques, and the best practices of creating modular components. Based on the analysis, a 3D first-person logic game was designed. This thesis presents the game design document and describes the creation of modular components in Blender with materials and some textures. The game was implemented in Unity and tested by seven users.

Keywords: 3D logic game, Unity, modular components, game design.

Abstrakt

Tato práce se zabývá analýzou herních principů logických her, analýzou celkového procesu návrhu levelů a modulárních komponent, k čemu slouží a jak se vytváří, a jmenuje dvě modelovací techniky. Na základě této analýzy byla navržena 3D logická hra z pohledu první osoby. V práci je obsažen její game design dokument a popis tvorby modulárních komponent v Blenderu, včetně jejich materiálů a některých textur. Výsledná hra byla naimplementována v Unity a otestována sedmi uživateli.

Klíčová slova: 3D logická hra, Unity, modulární komponenty, hení design.

Acknowledgements

I would like to express my deepest appreciation to my supervisor Ing. Ladislav Čmolík, Ph.D., for his great insights into the topic of this project and his valuable advice. I am also grateful to my family and friends who supported me throughout the whole period of writing this thesis. Special thanks to the testers who, voluntarily and without any claim for reward, immensely helped me with the development.

List of Figures

Contents

Chapter 1

Introduction

The gaming industry has evolved enormously since the release of the first game. It still keeps evolving and getting bigger every day, with an average of 28 games released on Steam each day last year [8]. It is also a very competitive industry because of the massive number of newly created games. The player may choose the game that intrigues them, and that seems to be the best one to play, leading to many games falling into oblivion. This puts pressure on game developers and designers to come up with new great ideas, such as an interesting storyline or new developing methods.

Creating a video game is a long process that consists of game design, creating a story, graphics, programming, sound, and more. It is not possible to focus only on one part when creating a game. Game development is a package of everything mentioned that all together influence the player's experience while playing the game.

There are engines that make development easier. These engines allow developers to create the whole game, including models, scripts, sounds, and animation while maintaining straightforward manipulation. There are many options from which to choose, which makes the decision harder. It is important to compare what the engine may offer with how easily it can be learned to use. To create the game, I use the Unity engine. Unity allows modifying a scene with preimplemented basic physics and a game loop. The developer's work is adding game objects and writing scripts that define the object's behavior and characteristics. Objects may affect other objects, which is again the developer's work to define how the objects are affected. The developer must also design levels in the right way to keep the player's desire to play the game. This includes creating exciting combinations of game elements.

Part of the work of developing a game is also creating game models that fit the game's theme. If the game is not visually pleasing, it does not interest any player, so this part is equally important as the programming part. To create models of the game elements, I use Blender, a software that enables creating 3D content easily importable in a .blend file to Unity.

In the thesis, I focus on creating a 3D logic game from a first-person view. The game is about a girl that has to escape a foreign world locked up in a series of rooms. Each room consists of a puzzle that needs to be solved to continue to the next room. The player controls the girl and her every move.

The following text is structured in a way that the first chapter focuses on the description of Unity in chapter 2 because the rest of the thesis is based on it. It is followed by research done on multiple topics in chapter 3 Game Design Analysis. The game design section analyzes the problematics of creating a game in general. Because the goal of this thesis is to create a puzzle game, this chapter particularly focuses on puzzle games. I look at the main principles of creating interesting puzzles and analyze a similar game Lara Croft Go according to these principles. In section Level Design, I describe all parts of the game design in general. I focus on suitable approaches to level design.

Based on the previous chapter's research results, I design a 3D logic game and write its

Game Design Document in chapter 4 Game Design Document. I mention the game's features, game concept, game control, game elements, and combinations of game elements that create the specific puzzle.

The following part of the thesis is about the creation of modular components. I talk about the modular level design, the representation of 3D objects in 3D graphics, and provide an overview of two practical modeling techniques useful for creating assets for this game in chapter 5 Asset Creation Analysis. I also write about creating materials that can be later used on game objects as well as texturing.

According to the analysis, I define the modular components necessary for this game and describe the overall process of the creation of game models in chapter 6 Creating Modular Components.

In chapter 7 Implementation, I briefly explain the utilization of every class. I also show an example of one puzzle, where I describe the correct walkthrough to solve the puzzle and leave the room.

In chapter 8 Testing are described two tests done with overall seven volunteers, which lead to altering the elements and game logic to fit the player's requirements.

This thesis aims to create 3 levels of a first-person 3D logic video game using the Unity engine and to create modular components and use them in the game.

Chapter 2

Unity Engine

For the implementation of this game, I decided to use the Unity Engine. In this chapter, I talk about its characteristics and show the basic technics to create a computer game using this engine. I use this platform because I am the most familiar with it. It is easy to comprehend with extensive documentation, and most importantly, it is free, even though there are also paid subscriptions.

The choice of the engine affects the process of the development of the game. For example, modeling and creating textures must be done with keeping in mind the engine's characteristics.

Unity [29] is a development platform for creating 2D and 3D multiplatform games and projects for film, animation, transportation, and simulation. It was launched in 2005 by Unity Technologies to make the development more accessible to more creators. Since then, many versions of Unity were released. (I use version 2020.1.10f1.). Unity has a built-in Visual Studio integration with C# API. Popular games using a version of the Unity engine are, for example, *Subway Surfers*, *7 Days to Die*, *Hearthstone*, *Ori and The Blind Forest*, *Cuphead*, *Iron Man VR*, and *Fall Guys: Ultimate Knockout*.

If not stated differently, this section is based on the Unity's User Manual [33].

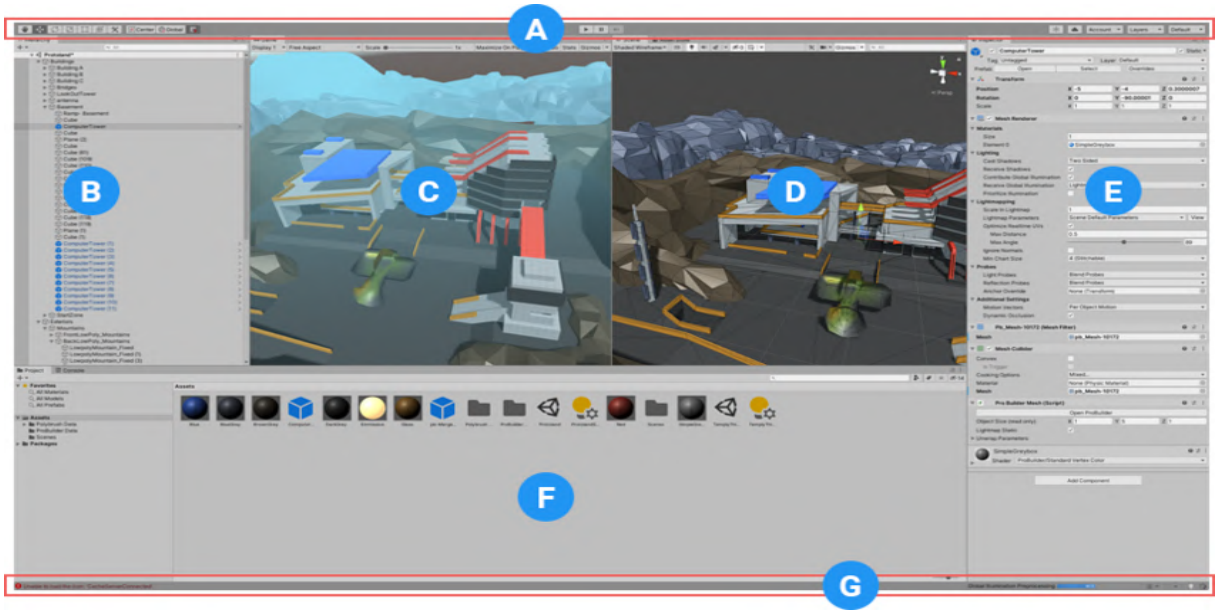


Figure 2.1: Unity's interface. A) The Toolbar, B) The Hierarchy window, C) The Game view, D) The Scene view, E) The Inspector Window, F) The Project window, G) The status bar

2.0.1 User's interface

Unity's interface consists of the toolbar in the upper part of the screen. Figure 2.1 shows the interface divided into seven parts. In the center is the current scene view. On the left is the Hierarchy Window, a hierarchical representation of all GameObjects in the scene. The hierarchy of GameObjects illustrates their parent-child relations. In the right part is the Inspector Window, which lists all properties of the selected GameObject. The Project Window displays the project's Assets, like scripts, models, or textures, in the bottom part.

2.0.2 Assets

Unity is based on the creation of scenes using assets. An asset represents any object, which can be used in the project. It may be a file created outside of Unity, like an audio file, an image, or a 3D model, or it may be a file created within Unity, such as ProBuilder Mesh, an Animator Controller, an Audio Mixer, or a Render Texture. Unity also allows downloading assets from its site Unity Asset Store from other creators. Unity allows using prefabs, which makes it easier to build the environment more efficiently. The developer can create new instances of the Prefab in the scene. Prefab's system allows confining and storing a GameObject complete with all its components, properties, and its children as a reusable asset.

Scenes

As already mentioned, Unity's main feature is managing scenes and their modifications. The scene is the most important part of the development in Unity as it is an asset that contains all or part of the game. It is where the developer puts the characters, creates the environment, or builds the obstacles. The game may be stored in one scene or scattered in multiple scenes, for example, one level = one scene. It is also the first thing that the developer sees when s/he creates a new project in Unity.

Game Objects

A game object represents any object currently in the scene, but it does not have any purpose on its own. To give the object a type of characteristics or behavior, it must include a component. A component is what describes the game object further and what even controls it. A game object can contain more components as one component may be assigned to more game objects. An inspector window shows how many components a game object has and what those are.

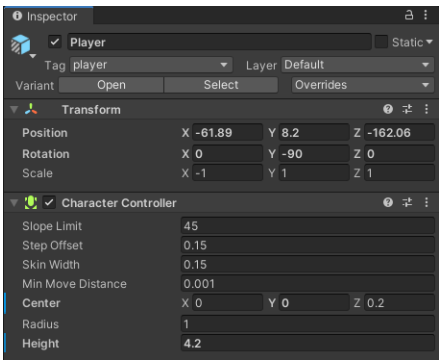


Figure 2.2: The Transform component in the Inspector window with the Character Controller component

When added to the scene, every game object has a Transform component assigned to it by default, which can be seen in Figure 2.2. The transform component specifies the game object's position and rotation relative to its parent. It also defines its size. Without the Transform component, the game object could not be placed in the scene because it would not carry any position information.

Other components can be assigned using the Add Component button. The components may be a material that sets the object's visual properties, color, metallicity of the object, or its smoothness. It also allows setting objects normal maps or height maps. Other components may be scripts or shaders, colliders, physics components like the Rigidbody component, or a mesh component that defines its shape, such as the Cube mesh, the Cylinder mesh, or the Sphere mesh.

The creation of game objects and their components As previously said, Unity allows importing 3D models. Unity cooperates with various modeling programs as Blender [5], a program used to create 3D models and animations. Models exported as .blend files must be put into the Asset folder of the project for Unity to register them. If the object is then modified in Blender and saved, Unity is able to load the modified object and apply the changes in the scene even when the editor is opened. Another way is to create objects in Unity by adding a new object into the scene.

When an object is added into the scene, it can be added beneath other objects as their child, or other objects may be added as its children, as shown in Figure 2.3, where the Canvas game object is a parent of Dialogue, Menu and Finish game objects. The Finish game object is inactive, while other objects are active. The game acts as the Finish object is not in the scene, because of its inactivity. The object can be activated via a script when needed during gameplay.

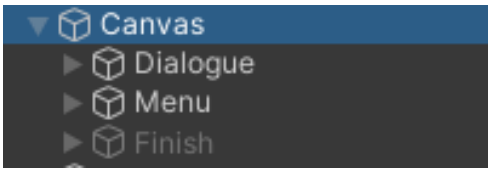


Figure 2.3: Parent-child relations between GameObjects in Unity

The added object can be manually rotated, translated, or scaled using tools from the toolbox, or these properties can be changed by changing numbers in the Transform component. After selecting its position, rotation, and size, the developer has to define the object's purpose in the scene and what it should do by attaching proper components.

2.0.3 Physics system and movement

Unity offers a built-in physics system that can be activated through the Character Controller component, Rigidbody, or Colliders. One of the main things the physics system provides and controls is the movement of game objects. Certain objects move, and others stay static. Unity offers three options for implementing the movement: the Rigidbody property, Character Controller property, and NavMesh Agent. It is also possible to manually set the position in every FixedUpdate by changing the Transform property, but it is not recommended due to objects not interacting with other objects.

Colliders

Colliders play a massive part in the physics system; they are necessary within all these components. Colliders replace the mesh of the object in collision computing. They are usually only approximations of the mesh of the object for easier calculating of the collisions. Unity provides primitive colliders as components, such as the Box Collider, Sphere Collider, and Capsule Collider, but sometimes, it is necessary to have more complex colliders that more correspond to the actual shape of the object. In that case, Unity offers the Mesh Collider, which matches the object's shape perfectly. However, the problem with this approach is that mesh colliders are more computationally requiring.

Colliders allow controlling if a collision occurs. If the IsTrigger property is enabled, the colliders do not act like solid objects and let another collider pass through. However, this encounter calls the OnTriggerEnter function, which can be implemented in the script.

Rigidbody and Character Controller

Both Rigidbody and CharacterController components are preimplemented parts of Unity.

CharacterController, as the green color shown in Figure 2.4, is a capsule-shaped collider whose movement can be easily controlled by a script. The controller then carries out the movement. It allows walking up the stairs, or slope climbing.

Rigidbody component is a component that enables the game object to be affected by forces and move in a realistic way.

CharacterController has a collider within, but for the Rigidbody component to be assigned, the collider on the object must be present, or it cannot be assigned. Both react to collisions, but Rigidbody puts its motion under Unity's physics engine's control while the CharacterController is not affected by forces and does not push other Rigidbodies away. CharacterController, on the other hand, allows easier movement, including slope climbing. It depends on the user's preferencies.

It can also be combined. For example, when the main character uses the Character Controller component but has to be under physics controls on various occasions, such as when a bomb goes off near the player. When it would be too difficult to implement all affecting forces in the script, the ragdoll effect should be used.

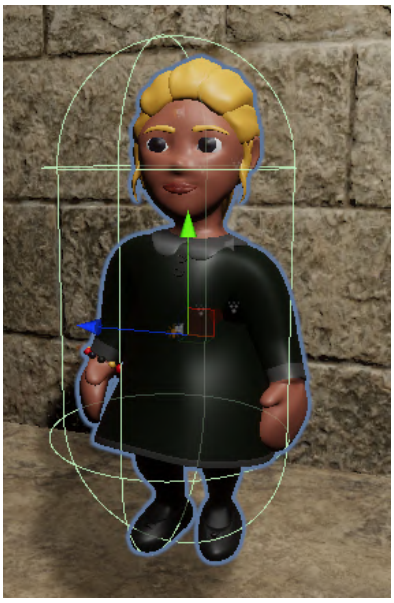


Figure 2.4: Cylindrical shape of the Character Controller's collider (light green color)

The Rigidbody component has a few properties that define how the object reacts to Unity's physics. The isKinematic property defines if forces, collisions, or joints affect or do not affect the rigid body. Kinematic rigid body is useful for a static object that is not affected by forces. After it is triggered, it deactivates the isKinematic property to move, and shortly after, it activates it back. The useGravity property defines if the rigid body is affected by gravity. It is useful to turn the gravity off if the object is upside down and is affected by user-implemented forces. Simulating gravity in the opposite direction is possible by calling the AddForce method of the Rigidbody component in the FixedUpdate method. A vector with zero in the x and z coordinates and 9.81 in the y coordinate must be sent to the AddForce method simulating the inverted gravity.

Interacting with other game objects

The Rigidbody object reacts to other objects with the right collider. If it bumps into a box only with a collider, it changes its direction but does not move the box. If the box also is a nonkinematic rigid body, then it is also affected by forces and moves in the same direction as the object. It results in the object slowing down. The object moves if another rigid body pushes it, or if it is moved via script.

For **CharacterController** to be able to push other objects, it must be implemented in a script. The function OnControllerColliderHit is called if the object's collider hits another collider. The method gives detail information about the collision. In this method, the velocity of the hitted object's collider's rigid body must be updated and the AddForce method must be called.

2.0.4 Audio

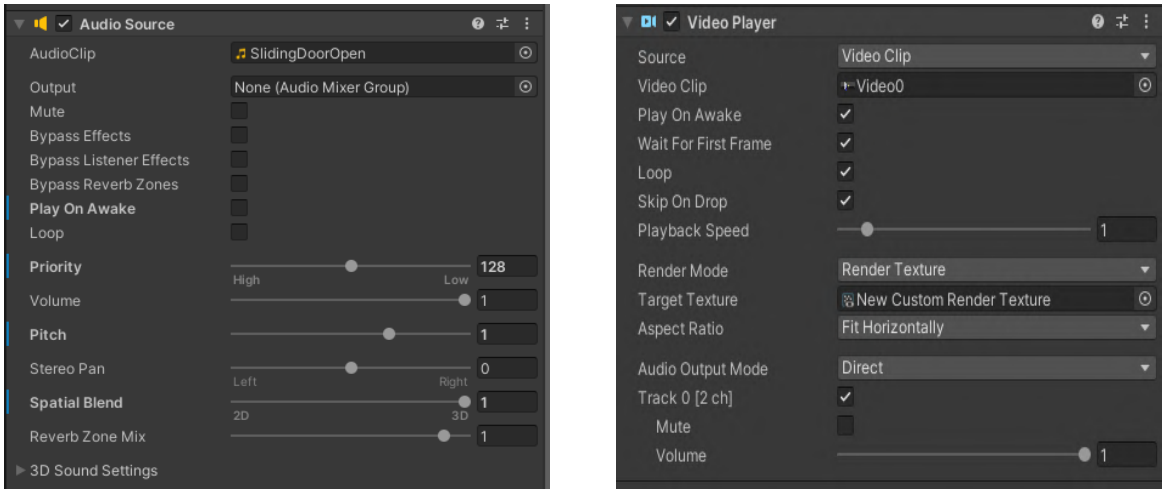
Unity provides 3D spatial sound. It stores the audio in the Audio Clip that can be used in the Audio Source component, shown in Figure 2.5 (a), which directly plays the sound in the scene. A few of the Audio Clip properties are the Play On Awake property, Loop property, which defines if the sound should be played in loops or stopped after it reaches the end of the sound.

Played audio can be processed by Audio Listener that is usually assigned to the Main Camera to create a realistic audio system. Or it can be processed by Audio Mixer, which allows mixing

multiple audio sources, applying effects to them, and performing mastering. There can be multiple Audio Mixers in the scene, whereas there can be only one Audio Listener.

2.0.5 Video

Unity provides the Video Clip asset, which stores the video data. It can be used in the Video Clip property of the Video Player component, which can be seen in Figure 2.5 (b). The video footage is then transferred into the Texture parameter of any component. Apart from the properties that were mentioned with the Audio Source component, the Video Player component also has the Render Mode property that defines how the video is rendered. The video can be rendered on Camera's near or far plane, using the Camera Near/Far Plane variable.



(a) The Audio Source component with Audio Clip assigned

(b) A Video Player component with Video Clip and Target Texture assigned

Figure 2.5: The Audio Source and the Video Player components

2.0.6 Animation

Unity Animation System allows animating any object in the scene, or controlling humanoid animations, including animating different parts with different animations. It also allows setting transitions and interactions between more animation clips. It also supports animations created outside Unity.

An object that should be animated must have the Animator Controller component assigned. Within the Animator Controller, there is a Controller property that requires the Animation Controller. Animation Controller controls the Animation Clips. This can be managed in the Animator window, which is shown in Figure 2.6.

Animation Clips support importing animations from other sources, or they can be created using the Unity animation system in Unity Animation Window, shown in Figure 2.7. Animation clips can store and animate the object's position, rotation, and scale, or the object's other properties such as material color, sound volume, and others. It can control the animation of properties of a script, such as float, integer, enum, vector, and Boolean variables. It can also animate the timing of calling functions.

Animator Controller allows switching animations when the defined variable changes. In Figure 2.6, it is the Speed variable. It can control if the character visually walks or runs based on the speed of the character. These transitions are enabled via State Machine.

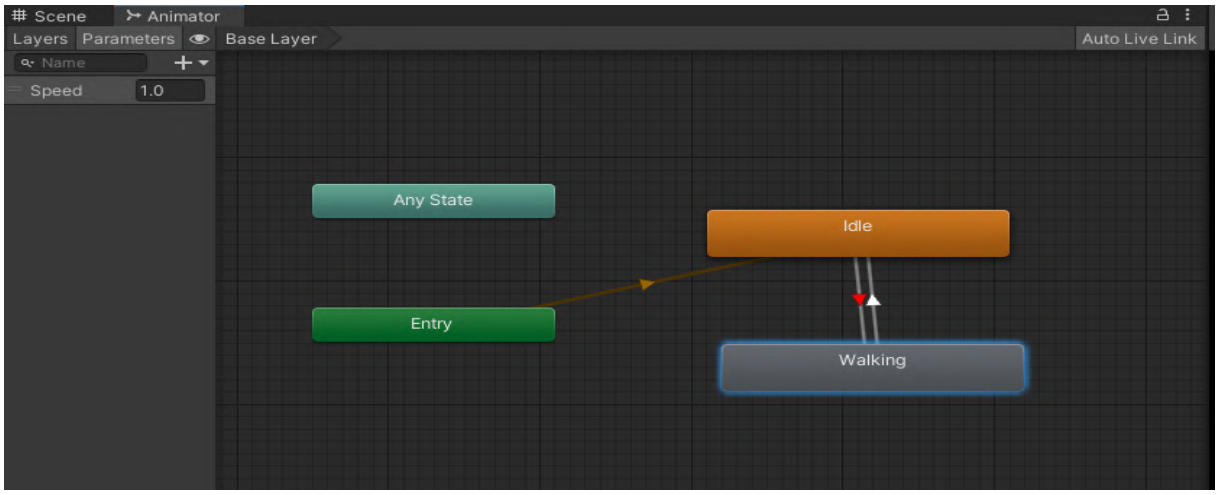


Figure 2.6: the Animator Window with states (Entry, Idle, Walking) and transitions between these states represented by arrows. In the left corner, there is a variable Speed that controls switching of the animations

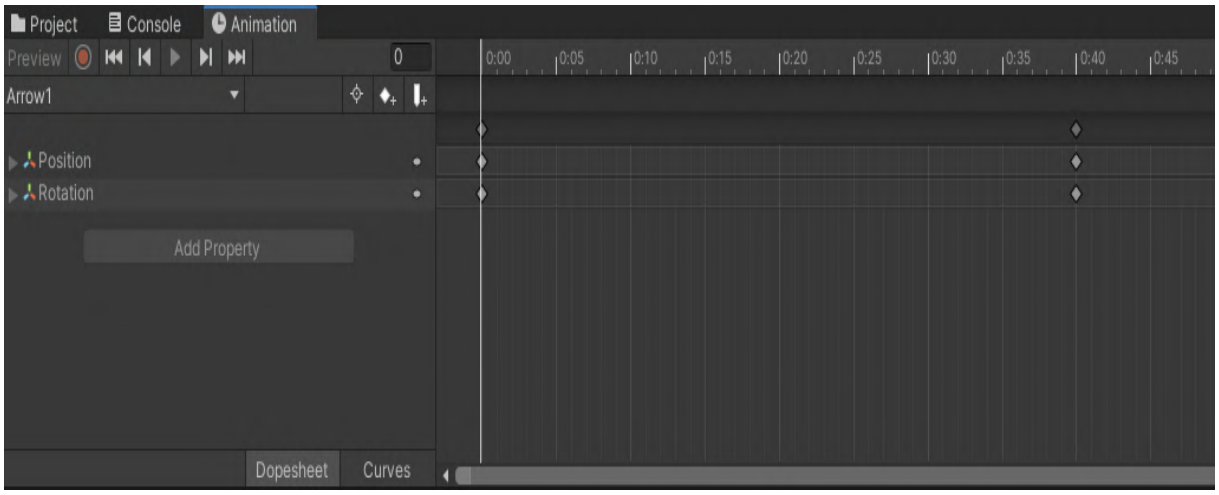


Figure 2.7: The Animation Window with animated object's position and rotation

Chapter 3

Game Design Analysis

This chapter focuses on designing a 3D puzzle game. At first, I describe the general process of designing a level, which helps setting a structure for my game's development. Then, I analyze the principles of creating a good puzzle game, which could be helpful when designing my own game. And finally, I look at one puzzle game where the player must interact with the environment and move objects to solve the puzzles, and analyze the game according to the principles presented.

3.1 Level Design

Level design is one of the most important parts of game development because the good level design may improve the whole gameplay, as much as poor level design may lead to chaotic or even boring gameplay.

Game level designers must put together a level with all components, puzzles, awards, and storytelling to create a gameplay that is interesting and fun to play. S/he must choose the right amount of everything to make a good game. *Level design is the process of creating playable content that serves as a medium of delivering gameplay and storytelling experiences to players.* Certain game designers create detailed levels with everything planned out, whereas others rely on procedural creation of levels and randomization, even though a human still must set some basics. [16]

3.1.1 Process of Level Design

Level design is different for every game because every game is different. Even though the level design differs and there is no general recipe on how to do it, there are several steps that may help with it. [16]

1. The premise
2. The sketch
3. GrayBoxing
4. Art implementing
5. Final polish

3.1.2 The premise

The premise gathers all the ideas that come up with the game. It sets up a high-level vision. It is expected that it changes during the game development even more, when written down long before actually creating the game. It should highlight the purpose of the game and may consist of several reference images. [16].

In the premise phase, the game designer must come up with a theme that the game will be set in. It affects everything in the game, from the main story to the creation of models.

3.1.3 The sketch

The purpose of the sketch is to revisit and expand on the plans before in-game implementation. The sketch can contain written design details, actual sketches, gameplay flow, or timeline. [16].

An essential part is the story of the game. The story usually improves the general playthrough. Certain games do not have a story and are still successful, such as *Pong!*, or in various games, for example in *Minecraft*, the story is created by the player. It depends on the genre of the game, but even a basic storytelling can have a considerable impact on how the player is drawn to the game. The story should be told to the player throughout the level from point to point. Otherwise, the game gets boring, and the player cannot evolve further.

There are many types of goals from which the developer may choose: explore goal, educate goal, moral goal, or goal of escape or survival. [25]

The designer must also think about what is the objective of the level. If the level is supposed to teach the player something, for example, the tutorial level that is supposed to show the player how to jump or show the basis of combat. The designer has to decide how the story is written into the game and how the player will be guided through the story. The storytelling part is essential as poor storytelling can break the whole feeling of the game and seem out of place. The designer has to decide which tool to use to tell the story.

The most popular ones are: [16]

Cutscenes.

Dialog, whether interactive or passive.

Direct and Indirect Gameplay. In direct gameplay, the story development is based on the player's actions; whether in the indirect gameplay, the player cannot affect the story. It is based on world changes or NPC[1] actions.

Narratio. An excellent example of narration in a game is *Divinity Original Sin II*.

Environmental storytelling

3.1.4 Grayboxing

The next part in game-level design is grayboxing. In Figure 3.1 (a) is shown the grayboxing version of *Uncharted 4*. Grayboxing is used to transfer the vision of the game into a simple playable version. It is called grayboxing for untextured, usually grey basic models, mostly boxes, that allow the player to test the gameplay in the early development stage. Based on the testing and feedback, the game design may change, and it often does. The most important thing before the visuals of the game is how fun actually the gameplay is. If it is no fun, then visuals such as lovely textures and detailed models cannot make up for it. [16]

It is good to validate the game content in this stage, verify the geometry, layout or sequence of events, set the base difficulty, place and configure enemies, puzzles and obstacles, cutscenes, and story scenarios and tutorials.

Art Implementation

After the grayboxing stage, it is time to focus on the game's environment and visual side. This means creating detailed geometry with textures, adding decoration and lighting to the scene, and creating special effects and animation. [16]

[1] Non-player character

(a) Grayboxing version [2] **(b)** Final version [10]

Figure 3.1: Level design in Uncharted 4

3.1.5 Final Polish

This is the last part of the process of level design. It can be set long after the heavy testing during the grayboxing phase and after all models have been created. As the name suggests, it focuses on the details and only smoothens the game. This includes fixing bugs, minor tweaks to the geometry, few dialogues or difficulty changes, and rewards adjustments. [16]

It is very important to plan the whole process beforehand to prevent chaotic changes that could lead to discontinuity.

The main workflow of developing this game starts by designing the game principles, defining the game elements, and planning several combinations of these elements. The premise and the sketch phase are generally covered in the game design document.

The next part consists of designing the modular components according to the game elements, followed by creating basic models to use them in the grayboxing stage with only a partial implementation, which should be part of the first test to determine if the game is fun to play and the logic is understandable.

The following part is detailing the models of the components and creating their materials and textures. The final part is building all levels using these modular components and finishing the whole game.

3.2 What should games provide

According to Raph Koster in A Theory of Fun for Game Design [15], games should provide a pleasurable learning experience. Koster mentions several ways that could fail this approach: [15]

- If the player finds out how the game works within the first five minutes of gameplay, s/he might dismiss the game as trivial with the words "Too easy".
- The player might find the game with a ton of depth to the possible permutations but conclude that these permutations are below their interest level. Something like memorizing the patterns is not worth the time.
- Or the player might not even see any pattern in the game, resulting in the player's boredom with saying, "This is too hard".
- The pacing of the unveiling of variations in the pattern might be too slow. The player might dismiss the game too early. "This is too easy now – it's repetitive".
- Or the variations might be unveiled too quickly, in which case the player might give up on the game due to losing control of the pattern. S/he says, "This got too hard, too fast".

- Or finally, the player might master everything in the pattern. There is no fun left in the game.

During game development, the developer needs to think about all these principles. For example, in each game level, adding something new that complicates the game and keeps the player thinking while, on the other hand, not making it too hard.

3.2.1 Puzzle games

In this section, I look at various ways how to ensure these previously mentioned points in puzzle games.

Puzzle games are a massive part of the game industry, as many players like the feeling of satisfaction when figuring out a mystery, a riddle, or a type of problem. My game is hugely based on puzzles, and that is why it is essential to know how to create the best puzzles that are favorite among the players.

If not mentioned differently, this part is based on the Art of Game Design by Jesse Schell [27]. Schell says that a puzzle is a game with a dominant strategy. It makes the player stop and think about the problem. Schell asks the question if the puzzles are dead. States that puzzles were very popular in the 1980s and 1990s, but nowadays, action-based games are trendy. However, mental challenges can add significant variety to an action-based game and that game designers know this. With experience, modern designers incorporate puzzles into the environment. For example in *Legends of Zelda: The Wind Waker*, where there are many puzzles, but the game smoothly integrates them into the environment.

At one point of the game, the player is confronted with a lava river. S/he has to figure out how to throw water jugs in the right pattern to cross the river, which is shown in Figure 3.2. In a dungeon where the doors are opened and closed by a complex series of switches, the player must figure out how to use the items found in the dungeon to flip the switches to get through all doors successfully. Enemies in the dungeon are paralyzed when light falls on them. To get the doors open, the player must lure the enemies to the right switches and then shoot flaming arrows near enough to paralyze them to keep the door open. All puzzles in the game are a natural part of the environment, and the goals of solving the puzzle are the player's avatar's direct goals.

Schell put together ten principles of creating a good puzzle that can improve the gameplay.

Figure 3.2: The player has to throw water jugs in the lava river to be able to cross it [11]

Make the goal easily understood

The player should know what s/he is supposed to do. Otherwise, s/he might quickly lose interest. The only possibility is when the player enjoys figuring what to do with the puzzle, but this could be only for "diehard" fans.

For example, in Hasbro's *Nemesis Factor*, which is an electronic puzzle with five buttons: red, green, blue, yellow, and orange, as shown in Figure 3.3, the player must light all buttons by pressing them in the right order and at the right time. It has 100 levels, and each level is different. Sometimes, the buttons play sounds, musical notes, or words. [22]

This game did not sell well, because it violated the first principle, as the goal was not clear. It is hard to predict how to interact with the game or what to do to achieve the next level. Puzzle lovers must admire this game, but it seems rather frustrating for the general public because it gives only a little feedback, and the player does not know if s/he is on the right track.

Figure 3.3: Nemesis Factor [22]

Make it easy to get started

How certain puzzles look makes it obvious how the player should start. For example, in Sam Loyd's *15 Puzzle*, where the player must order 15 numbers on a 4x4 table with one free slot that enables moving with other cubes, the player knows the goal and where to start at first glance.

Give a sense of progress

Schell asks a question, "What is the difference between a puzzle and a riddle?" and says that a riddle is just a question that requests an answer, and a puzzle also demands an answer, but it gives the player the feel of getting closer to the solution.

Players like this sense of progress - it gives them the hope of actually resolving the puzzle, which gives them the drive to solving. Riddles, on the other hand, make the player think and guess the answer. S/he might get it wrong or right. In early adventure games, riddles were often used due to easy implementation, but their solving is sometimes too frustrating that they are mostly absent from modern games. *Rubik's cube* success is also based on the sense of progress it gives, so the players persistently try to solve it.

Figure 3.4: Rubik's cube - Faster Action No Sticker Cube [26]

Give a sense of solvability

If the player might think that the puzzle is unsolvable, s/he might get afraid that just wastes time and give up. On the other hand, when sold, the Rubik's cube is already in the solved state. The player then must twist it herself/himself a couple of times. It all makes the cube visibly solvable for the player

Increase difficulty gradually

One way of ensuring that the difficulty increases gradually is to give the player control of the order of steps in the puzzle. In the crossword puzzle, the player usually first chooses the easier

words and, after writing them, moves towards the harder ones. Giving the player, this kind of choice is called parallelism.

Hints extend interest

If the player is stuck and does not know how to solve the puzzle, gets frustrated, and even wants to give up, a well-targeted hint may restore her/his hope, curiosity, and drive. It may cheapen the experience, but solving the puzzle with a clue is better than not solving it at all. Even though Harbro's *Nemesis Factor* was not that successful, it did have a great hint system. It featured a hint button that played a one- or two-word hint about the current puzzle after pressing. Pushing it more times gave a bit more complex hint, but it also punished the player by reducing their points. Nowadays, it is not difficult to find the answer on the Internet, but for the player, it is more enjoyable to solve the puzzle by herself/himself rather than getting the answer from someone else.

Parallelism lets the player rest

If the player cannot solve the puzzle, s/he may get frustrated with being unable to make progress and give up. Another way to prevent this is to give the player an option to move to a different puzzle. In that case, if the player gets tired of one puzzle, s/he can go off and try another one. The player can take a break from solving the first one, and with time, the solution may come to her/his head. *Crossword* or *Sudoku* do this naturally, but videogames may do it too. It is recommended to give the player two or more parallel challenges at once, so the player gets less likely frustrated with the game.

Pyramid structure extends interest

A pyramid puzzle structure is a series of small puzzles that give clues to a larger puzzle. An example is the *Jumble* scrambled word game that appears in newspapers. An example of *Jumble* is shown in Figure 3.5. This game gradually increases the difficulty, and by unscrambling all words that lead to unscrambling phrases, the game as its goal gives the player the punch line of the presented joke. Thus, in the pyramid structure, pieces of the puzzle feed into a singular challenge at the end, a hierarchy of challenging puzzle elements gradually leads to a final challenge, and the challenge at the top of the pyramid must be interesting, compelling, and clear.

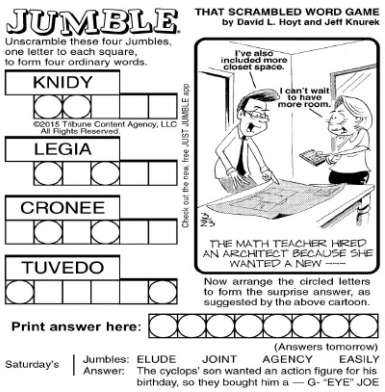


Figure 3.5: Jumble [31]

Give the answer!

For the player, it is also enjoyable seeing/reading the answer. A good example may be detective books, which are just puzzles in a book form. In the books, the reader may predict who the killer is, but if not, the killer's final reveal is surprising, which is just as pleasurable, or even more pleasurable, as figuring it out alone. The best approach is to give the player a way to determine the answers to the puzzles from within the game.

Perceptual Shifts are a Double-Edged Sword

The puzzles that involve a perceptual shift where you get them or do not are a problematic double-edged sword. For example, a puzzle, where the user has to move one of the matchsticks, usually forms a pattern to create a new one. If the player can make the perceptual shift, s/he receives great pleasure and solves the puzzle, but otherwise, they get nothing. These puzzles do not get gradually more difficult nor create a sense of progress. They are like riddles and should not be used in a video game, where the player expects a type of progress.

3.3 Similar game

During the game development process, the developer should look around the gaming market to see if the game s/he wants to create has not already been created. If not, it is good to look at similar games and get inspiration from them, as well as learn from their mistakes.

In this part, I talk about a similar game *Lara Croft Go*. Even though it is a mobile platform game, it has similar features to a game I would like to create, like puzzles based on changing the environment and moving with objects to find a path. Furthermore, it was created using the Unity engine.

3.3.1 Lara Croft: Go

It is a turn-based adventure video game developed by Square Enix Montreal [12]. The player must avoid obstacles and manipulate the environment. Levels are composed of lines and nodes in which the player can move. Every turn ends in a node. The player and the environment take turns. The player plays while the environment sleeps and otherwise. The goal is to find a path further without dying and complete all levels. Additional challenges are finding relics that are hidden around the ruins.

The game received mostly positive reviews. Lara Croft's well-known character probably helped with the game's promotion, but its gameplay and all those puzzles are what makes the game that successful. It even won The Game Award for Best Mobile/Handheld Game of 2015.

Figure 3.6: The player throwing a spear at the snake in Lara Croft Go [12]

The game is not all puzzles. It offers a storyline that the player needs to follow. The player explores the ruins of an ancient civilization where s/he discovers secrets and uncovers the Queen of Venom's myth. The puzzles become more complex and get more complicated on every level. New types of deadly animals that move in different patterns are added in further levels, pickable

objects such as a spear that can take down those animals from a distance, as shown in Figure 3.6, arrows fired from walls, and other game mechanics keep the player thinking and not getting bored. However, if the player is stuck, the game offers purchasable hints that help the player solve the puzzle and continue with the game.

Another principle of a good puzzle this game provides is the pyramid structure. For example, the player needs to collect three keys to open a massive door that enables the player to continue with the story. S/he must collect them in three different parts of the ruins with their own puzzles that need to be solved to get the keys.

Thanks to this game's popularity, it is not hard to find walkthrough guides on the Internet, showing exactly how the player must proceed.

In summary, this game meets many of those principles that Schell mentions. It offers a hint system. It is clear what the goal is and how to achieve it. It increases the difficulty of the puzzles with new game mechanics added to the game in further levels. The player must solve minor puzzles to be able to solve the bigger ones. The player sees on a map how s/he progresses in the game and is closer to collecting the runic and escaping the temple and having side challenges of collecting hidden relics. Besides, it is also visually pleasing, which gives the player a lovely feeling just looking at the game.

This game could serve as an inspiration on how to increase the difficulty, such as by adding more elements, or how to interact with the environment, for example, that the player must move certain objects to solve the puzzle.

A good puzzle game should include some of these principles, so it is vital to have these principles at the back of one's mind when designing a puzzle game. A few of them could be more fitting to my game than the others, for example, having the goal easily understood, giving the player a sense of progress, parallelism, or gradually increasing the difficulty.

Chapter 4

Game Design Document

The game design document describes the game in detail and serves as a set of ideas and instructions for the developers on how the game should be created.

The game was designed and its game design document was written based on the analysis done in chapter 3 Game Design Analysis and altered according to the test results described in chapter 8 Testing.

According to the analysis done in section 3.1 Level Design, the game design document specifies the game details, the main theme, the timeline, and the goal of the game. It includes sketches of the game, such as sketches of the game elements, the world, or certain puzzles. In the level design section of the game design document, each level is presented, its goal is defined, and three puzzle rooms are described.

The game design document has been altered and rewritten throughout the game development process many times, and the first version remarkably differs from the last one. This thesis includes the last version.

4.1 Basic Characteristics

It is a 3D fantasy logic game from a first-person perspective. The player must solve room puzzles to escape and find a way to return home. This game is inspired by many logic games created before, such as *Lara Croft Go* or *The Portal*. In these games, the player must move certain objects to trigger an action, which helps them progress with the puzzle. As *The Portal* has portals, this game also has a unique touch to itself, which is the presence of the parallel world.

The game is situated in an old castle, which sets the game's main theme. This influences the game models as well as the game logic. As already mentioned, it is a fantasy game, which means that the objects can be addressed or affected by various magical powers, such as crystals. The player walks through the rooms and solves puzzles along the way. The rooms are usually small, but there can be bigger ones. The playthrough is usually linear, which means the player must solve the puzzle in the room to get access to the next one. According to the analysis made in section 3 Game Design Analysis, it is advised to give the player, in case of being stuck, an option to try to solve another room and return to this one later. This leads to, from time to time, having two parallel rooms, where the player can choose which room s/he tries to solve. After finding three crystals that are located at the end of every level (including the tutorial level), the game ends with congratulations. The three crystals represent the role giving the player a sense of progress.

The game consists of three levels: the tutorial level, the first, and the second level, although more levels would be welcome in future development. Puzzles and levels are discussed later in the chapter. Each level consists of 7-10 rooms, which means the whole game comprises around 25-30 rooms. The rooms start as easier ones, where the goal is to push one crate to a pressure

plate, which opens the door to the next room, and as the game progress, the player gets to more complex ones, which means that the difficulty gradually increases. The levels are designed in a way that the player does not have to change worlds and try out every element to know what to do. The floor is transparent, which means s/he can see almost everything from the other world and plan the whole walkthrough. This means that the goal is easily understood.

4.1.1 The story

The player is a young girl Sammy who gets into a different world where reality can be tricky. She wakes up locked up in a cell in a totally unknown environment. She discovers an escape route, where she finds out she is in a mansion full of puzzles she has to solve to find three crystals to unlock the way home.

4.1.2 Movement

The movement and camera placement are similar as in *Minecraft*, as shown in Figure 4.1.

The player moves in four directions. Using WASD or arrow keys moves the player's body intuitively in a particular direction, whether the mouse movement rotates the body horizontally and vertically. By pressing CTRL, the player can crouch. By pressing the space bar key, the player jumps.

The Camera is placed at the front of the character's head, which means the player sees what the character sees. Again similar to *Minecraft* (Figure 4.1).

Figure 4.1: Screenshot of Minecraft

4.1.3 UI

Mouse Pointer

The pointer is locked in the center of the screen unless the menu or dialogue is opened. Then it is unlocked, and the camera does not move when the pointer moves. This enables picking an option in the menu or closing the dialogue using a Continue button.

Menu

A menu opens when the player presses the escape key. It also stops the game, and the player cannot move nor rotate the camera. S/he can move the mouse pointer and select an option in

the menu or close the menu by pressing the escape key again.

The menu contains a continue button that returns the player to the game. The menu disappears.

The load button loads the game from the last save. The save was performed when entering a new room.

The exit button quits the game.

Dialogue

A dialogue is triggered when the player enters a specific collider. The dialogue slips down and writes predefined sentences to the monitor in a specific area. There is a continue button that when pressed, displays the following sentences.

The dialogue is shown only when the game must inform the player about something important, mostly during the tutorial level, showing the player how the game works.

During the first testing described more in detail in chapter 8 Testing, it was suggested that the text was too long, resulting in the player skipping it. This led to including a video that illustrates what the player should do.

The dialogue slides upwards after all sentences were written.

4.1.4 Game saving

The game saves automatically at checkpoints right after stepping into a new room. The player cannot save by herself/himself because s/he could rewrite the last save. It could cause irreversibility of the solution in a way s/he would not be able to reload the room and start again.

4.1.5 Main character

Table 4.1: The main character's properties (the values are exactly defined for Unity)

Name	Value
playerSpped	8.0
crounchSpeed	2.0
jumpHeight	1.0
pushPower	2.0
playerheight	4.2
crounchHeight	2.0

As already mentioned, the character is a girl called Sammy. Even though it does not affect the game in any way, it is better to determine the gender and name to connect the player to the story.

The character has a few specifications that define her and the gameplay. These properties are shown in Table 4.1.

4.1.6 The Parallel World

The player enters a room in which s/he has to find a way out. The rooms are connected. When the player leaves the room while being in the parallel world, s/he enters the next room in the parallel world and otherwise. Certain rooms are connected more than just by doors. Then the player must return to these rooms.

The player must use both mirroring worlds and objects to crack the puzzle. The most important part of it is the parallelism of the world.

The parallel world is an upside-down world almost identical to the normal world, which is visualised in Figure 4.3. It is the main feature of the game, and all puzzles are based on it. By entering a portal, the player starts a rotation that switches both worlds.

Because I encountered during the first testing as written in chapter 8 Testing with remarks about differentiating and recognizing in which world the player currently is, both worlds are now differentiated by light. It is the typical daylight in the up world, while in the parallel world, the light is blue. Moreover, the parallel world and the up world distinguish by the possible actions. In the up world, it is impossible to move a mirroring stone (this means that the stone also projects to the parallel world) and can only be moved in the parallel world.

4.1.7 Portals

A portal enables the player to switch worlds and move to the opposite one. It is not limited to how many times the player can step into the portal and rotate worlds.

A portal consists of two pillars that are easily recognizable in the room. They have a space between them where the player walks to trigger the portal's collider, as shown in Figure 4.2. The space between them is distinguished by a barrier that tells the player where exactly the portal's collider starts. When the player touches this barrier, the rotating animation starts, and the worlds change.

However, the portal may be inaccessible due to an immovable object in front of or behind the portal. The player has to think about a different approach.

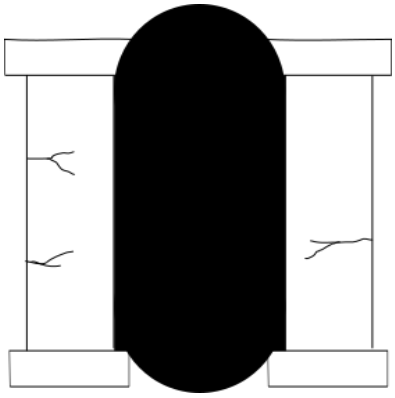


Figure 4.2: A portal

Rotating animation

The rotating animation is started when the player enters the collider placed between the pillars, as illustrated in Figure 4.3. The player movement is then stopped as well as the camera movement.

Both worlds are rotated around the player's camera, which results in the player staying in one place during this animation, but everything else changes its position, including the portals. That means that both rooms and all objects within the room rotate around the Main Camera for 180 degrees. The only object that is left out of the rotation is the player character with its children.

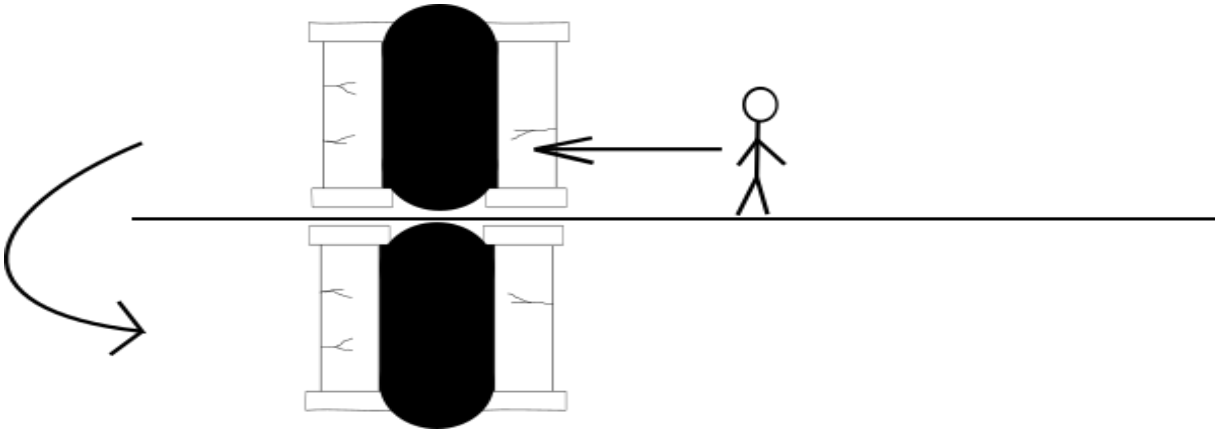


Figure 4.3: An example of the up world (up) and the parallel world (bottom) that mirrors the up world. The character must enter a portal to rotate the worlds according to the arrow

At the same time, the camera moves slightly backward at the start of the animation so that

the player is watching the back of the character, which leads to being better oriented in the space and not getting confused.

The whole rotation is about three seconds long, and after the rotation ends, the camera smoothly returns back to the front of the character's face.

4.1.8 Buttons

Buttons may be on a wall at any height. They may even be so high that the player cannot reach it from the ground without any box to jump on.

A button opens a door, opens a barrier, or brings another object to the scene. The button click is signalized by a sound and an animation that moves the middle part of the button back or forward. Button also shines with color that the object it affects shines so that it would be clear what it does even without trying it.

The button clicks when the player points the mouse at the center part and is close enough to reach the button. In other words, the distance between the player's camera and the button must be shorter than 7.

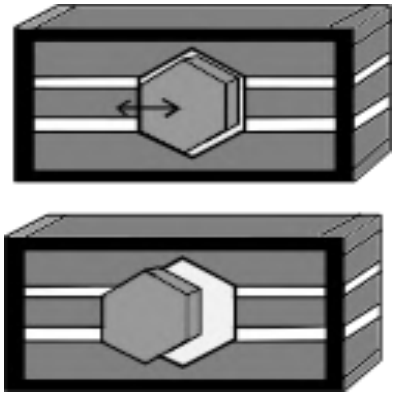


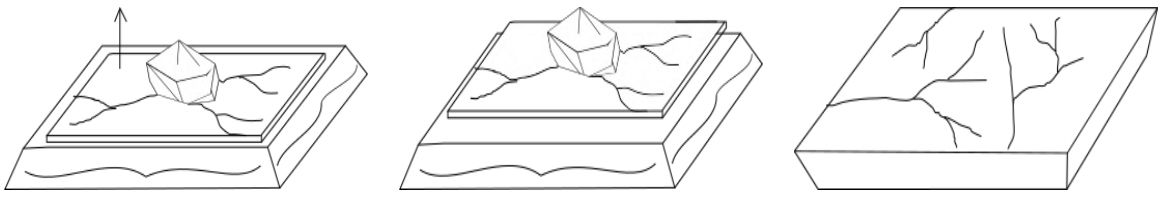
Figure 4.4: Pressed button (up) and open button (bottom)

4.1.9 Pressure plates

The pressure plate gets triggered if the player, AI, or any object gets on the plate. If the object enters the pressure plate's collider and is the first object to enter this area, the plate gets pressed. The middle part of the pressure plate then smoothly moves down and comes back up when released. Figure 4.5 (a) and (b) shows both of these stages of the pressure plate. The pressure plate has an overview of the objects that are currently inside its collider so that it does not get triggered every time an object enters its collider when another object already keeps the plate pressed.

The plate has a crystal built inside, as shown in Figure 4.5 (a), that shines the color of the object that it controls. It is usually another crystal of a door, which signalizes if the door is opened.

When triggered, the plate makes a sound that lets the player know that it was triggered. Moreover, the color of the crystal changes as well as the color of the object it affects. It stops shining, which leads to being painted in a darker color that does not have the bloom effect.



(a) A pressed pressure plate with a crystal in the middle

(b) An open pressure plate

(c) The bottom part of a pressure plate with crystal veins on it

Figure 4.5: The top and bottom parts of a pressure plate

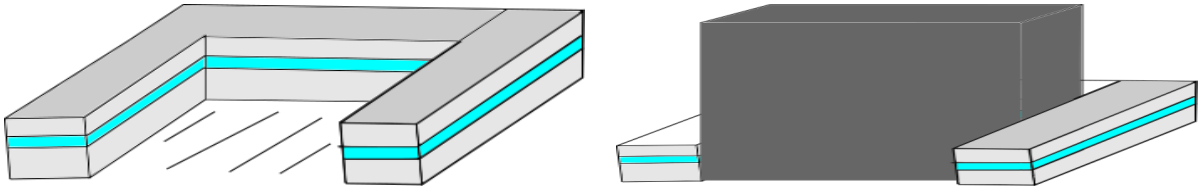
It is essential to let the player know what the pressure plate controls from the other world than it is placed in. This means that when a pressure plate is placed in the up world, but the player is currently in the parallel world, the player does not see the crystal that shows what object the plate affects. This would lead to confusion about the purpose of the plate, and

the player would be forced to change worlds to see what the plate actually controls. For this problem, the plate has on the bottom side cracks that are filled with crystal veins that behave the same way as the crystal. These crystal veins can be seen in Figure 4.5 (c).

4.1.10 Lift

The lift is used to lift the player. Lift is a block that grows upwards when activated, resulting in lifting objects that are placed upon him, which can be the player or a movable object. It usually mirrors to the other world, which means that also a mirroring stone can be lifted. It would not make sense to lift a stone only in one world. The rooms are designed such that the mirroring stone can never be moved to the lift if the lift does not mirror.

There is a short barrier around the lift that shows where the player should step to be lifted, as visualised in Figure 4.6 (a). Around the barrier, there is a built-in line that shines. A lift can be activated by a button or by a pressure plate.



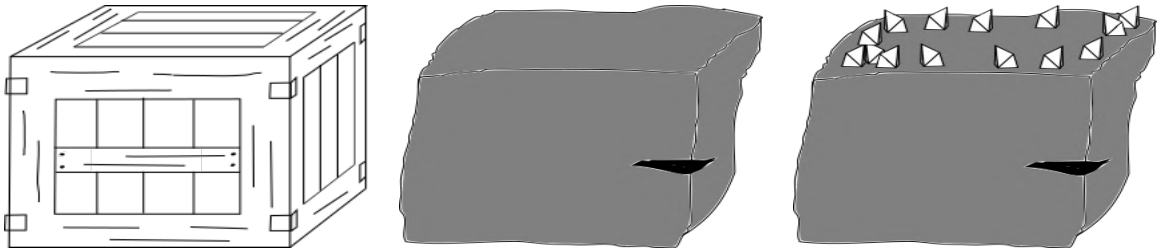
(a) The lift barrier with the lift being down **(b)** An activated lift

Figure 4.6: A lift

4.1.11 Movable objects

Crates

The crate is a block that can be moved or jumped on. Figure 4.7 (a) shows the design of the crate. The crate can be moved by the player or by other movable objects. This is the only interactable object that does not have any crystal on it because it is unnecessary. Crates never mirror to the other world and can be moved intuitively in the world they are placed. This means that if the crate is in the up world as well as the player is in the up world, the player can move the crate. On the other hand, suppose the player is in the parallel world, then the crate cannot be moved by the player directly. However, the player can move the crate indirectly by moving the mirroring stone, which then moves the crate.



(a) A crate **(b)** A stone in the up world **(c)** A stone in the parallel world with crystals on top

Figure 4.7: Movable objects

Mirroring stones

The mirroring stone mirrors to the other world. With its mirroring part, they share the same local position, which means that when the up world version moves, the parallel world also moves. When, for example, the player moves the parallel world stone, but the up world stone has a barrier in front of it in the direction of the push, none of them move.

The player can move only the parallel world stone. When trying to move the up-world version, nothing happens. This also led to minor confusion during testing 8 Testing. The testers were not always sure if they can move the stone. This resulted in the parallel world version obtaining a number of tiny crystals on top of it, as shown in Figure 4.7 (c). Therefore, when the player can see these crystals, s/he can also move the stone.

4.1.12 Door

Doors separate two rooms from each other and don't allow the player to continue until he has solved the puzzle in the current room. They usually open by smoothly sliding upwards and close by smoothly sliding downwards. The opening and closing are signalized by a sound effect that the player knows the door opened without looking at it.

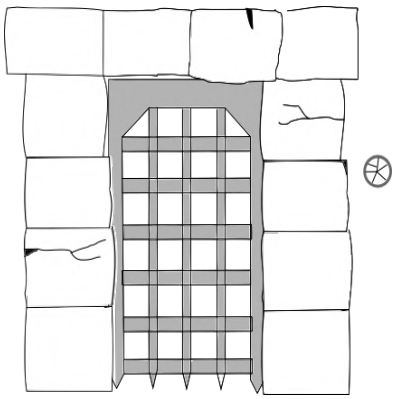


Figure 4.8: Closed door with a crystal on the right. One crystal = one object that alone opens the door

A door can be opened by one or more pressure plates or one or more buttons. It can also be a combination of both. This is shown to the player by the crystals that are next to the door. There is always one or more crystals placed next to a door. The crystals have different colors because the controlling objects have different colors. For example, if there are two pressure plates that together open a door, one pressure plate has its crystal blue and the other one red. The crystals next to the door are red and blue. Suppose the pressure plate with the red crystal is pressed, then its crystal and the red crystal next to the door stop shining. The door opens only partly and closes again. If the other plate is also pressed, then the door opens fully.

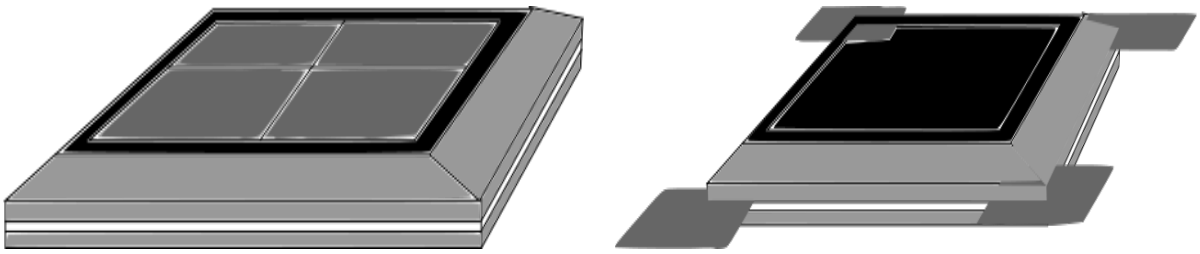
4.1.13 Teleporting plates

The teleporting plate teleports an object from one world to the other. The teleporting plate has a center part made up of four blocks that open along a diagonal. Figure 4.9 shows the teleporting plate with a closed and opened center part. It also has a white blooming part around the whole object.

It has to be activated usually by a nearby button that also shines with a white color, which signalizes that the button controls specifically the teleporting plate. There would never be two teleporting plates in one room, which means that the white color is always associated with the teleporting plate.

When opened and a crate is placed on top of those blocks, the crate falls through, particularly through a teleporting plate's collider, which changes its placement into the other world and switches its gravity settings.

The player cannot fall through the teleporting plate, only the crate can. The mirroring stones do not fall through because they put pressure on each other in different directions, which means they do not move at all.



(a) Closed teleporting plate with four blocks in the middle

(b) Open teleporting plate with four blocks that moved in the opposite direction along a diagonal

Figure 4.9: Teleporting plate

4.1.14 Moving platforms

The moving platform floats in a specified direction. It can move simultaneously or be triggered by a button that, after pressed, sets it in motion. A moving platform can be used for carrying a player or any movable object around. In the middle, there is again a shining part.

The platform transports movable objects, but the player has to walk to be moved.

The moving platform moves for a defined distance, and after reaching it, it stops.

4.1.15 Creating boxes

The creating box creates a crate inside itself. It is usually placed in the air, which means that the crate falls down after creation. A button or a pressure plate may activate it. It can also be activated when the plate is released. The crate is created from a prefab, which means it has every characteristic as a standard crate has.

At the top part, there is a shining block of a specified color.

4.1.16 Final crystals

There are three final crystals in the last room of every level. The final crystal has a collider that when triggered, disables the crystal and enables the one in the middle room. They all have different colors and are situated on the floor of the last room of the level.

4.1.17 Barriers

Glass barriers

Glass barriers are colliders with a rigid body that do not allow an object or the player to move in a specific direction. The barrier is a block that ends with a line that shines. The bloom color is usually pink, but when there are more barriers in one room, the color can be changed to whatever color is left. Glass barriers can be like doors, opened and closed. They can also be stopped from fully closing by a movable object that is under the barrier.

Disappearing barriers

Disappearing barriers are the only objects that disappear and reappear when an object controlling them is triggered.

4.1.18 Mirroring and non-mirroring objects

As already said, certain objects mirror to the other world, while others do not. Their position and their state are shared and reflected. It means that if a mirroring object moves or is, for example, destroyed in one world, it also moves or is destroyed in the other one. It can be a

mirroring stone, a pressure plate, a barrier, a lift, or a button. Portals always mirror so that a player can teleport to the other world, but it cannot be moved or destroyed.

The mirroring pressure plate is always pressed when its mirroring version is pressed. However, they can affect a different object. When they do, it is an object that also mirrors, which means that the plate affects the version that is placed in the same world.

A barrier is opened/destroyed when its mirroring version is opened/destroyed.

A lift is activated when the mirroring version is activated, as well as a button is clicked when the mirroring button is clicked.

Distinguishing which objects mirror and which don't can be easily done according to a simple rule. If these objects appear in both worlds in the same local position according to the room in which they are placed, they mirror. Although, this rule cannot be applied to crates. Two crates can be placed on the same local position in different worlds, but it does not mean that they mirror because crates never mirror.

Objects that never mirror are crates and teleporting plates because they lie between both worlds and interfere with them.

4.1.19 Changing levels

The levels can be changed from the middle room. There are two pressure plates that control specific doors that enable the player to enter the level. The plate's colliders are disabled until the player obtains a final crystal connected to the plate. Meaning if the player obtains the first crystal, which is the red one, the collider of the pressure plate with the red crystal enables, and the pressure plate is now usable. If pressed, it opens a special door with crystal veins along the surface that leads to the first level, and the door to the tutorial level closes. To save computing power, all rooms but the first level rooms are inactivated.

4.2 Combinations of game elements

Certain elements affect others. Pressing a button may open a door, stepping on a pressure plate may reveal a vital object necessary for unlocking the main door.

These combinations are crucial in solving the puzzle and creating a room with difficulty based on how far in the game it is. At the beginning of the game, the player has to at first get the feel of these combinations and how the objects react to each other. On the other hand, with each room, the difficulty increases in a way the game does not get boring.

Basic combinations: Crates can be placed on the pressure plates, or if something is too high for the player to reach it, the player can move the crate and jump on it, which gets him high enough to reach it.

A button may open a barrier or a door. It can also activate the lift or the teleporter plate, which then throws a crate into the other world.

A button or a pressure plate may open a barrier but at the same time, close another one.

Mirroring stones may be used to move an unreachable create that is, for example, blocked by a partly open barrier that is too low for the player to get under but high enough for the crate and stone. Stones can also press plates as well as be jumped on to click a button that sits too high on the wall.

Crates and stones can also block a glass barrier falling entirely on the ground.

A button may activate a lift that can be used to get the player somewhere high, for example, on a platform.

4.2.1 Level design

As already mentioned, there are three levels. The player starts in the tutorial level, from which s/he gets to the essential room in the game, the Middle room, from which can the player go to every level. Although, when the player gets to the middle room, s/he has already passed the tutorial, which means that returning to the tutorial level would be useless. The player also teleports to this room using a special portal in the last room of the first and the second level.

As mentioned in 3 Game Design Analysis chapter, it is recommended to show the player their progress. For this purpose, the final crystals were added.

The currently obtained crystals appear in the middle room.

When a final crystal is obtained and appears in the middle room, the specific pressure plate connected to this crystal is activated. The newly activated pressure plate then can open the door to the next level.

If the obtained crystal is the third one, a barrier in the center of the middle room is activated. When the player enters the barrier, the game ends.

In the next part, I talk about every level, what new elements they bring to the game, and present a few examples of the puzzle rooms in each level. Because there are overall around 25-30 rooms, I don't describe every room in the game.

The tutorial level

The first room introduces the game to the player, shows the game controls, and the player is taken through the next few rooms via the dialogue. The player does not spend much time in the first rooms, because their role is to teach the player how to move, show the essential game elements, and how to combine and use them in the correct order to solve the puzzle and move to the following room.

The tutorial level introduces portals, mirroring stones and non-mirroring crates, pressure plates, the disappearing barrier, and a possibility of crouching. Other mechanics are presented in further levels to make the puzzles more complex and more complicated to stick to the principles described in the 3 Game Design Analysis chapter.

Room examples

In the **first room**, the player has to move a stone in the parallel world to move the mirroring stone, which is illustrated in Figure 4.10. The player has to push it on the pressure plate that opens the door to the next room. The dialogue helps her/him and, through a video, shows what to do. The player enters the room in the up world and sees one stone in the center of the room. Next to it is a pressure plate that must be triggered to open the door. If s/he tries to move the box in the up world, s/he fails. The player has to enter the portal, which teleports her/him to the other world, where s/he can finally move the stone, which also moves in the up world. The player has to push the stone to the position where the mirroring stone gets on the pressure plate. The door in the up world opens when the plate is activated.

Figure 4.11 shows **The second room**. The second room introduces the non-mirroring crate and the ability to pull. The up world consists of one pressure plate, one crate, and a portal. The parallel world consists of these objects plus a door with two crystals, suggesting that the two pressure plates together can open the door. The player enters the room in the up world, although s/he leaves it in the parallel one. At first, s/he must pull a crate in front of the portal to be even able to change worlds. Then s/he must push it on the pressure plate, change worlds, and push the crate from the parallel world on the parallel's world pressure plate.

The seventh room, which is also the last room of the tutorial level, is entered from the parallel world. As shown in Figure 4.12, there are 3 mirroring stones, each of a different size. There is also a platform that leads to an open door in each world, but it seems that the platform

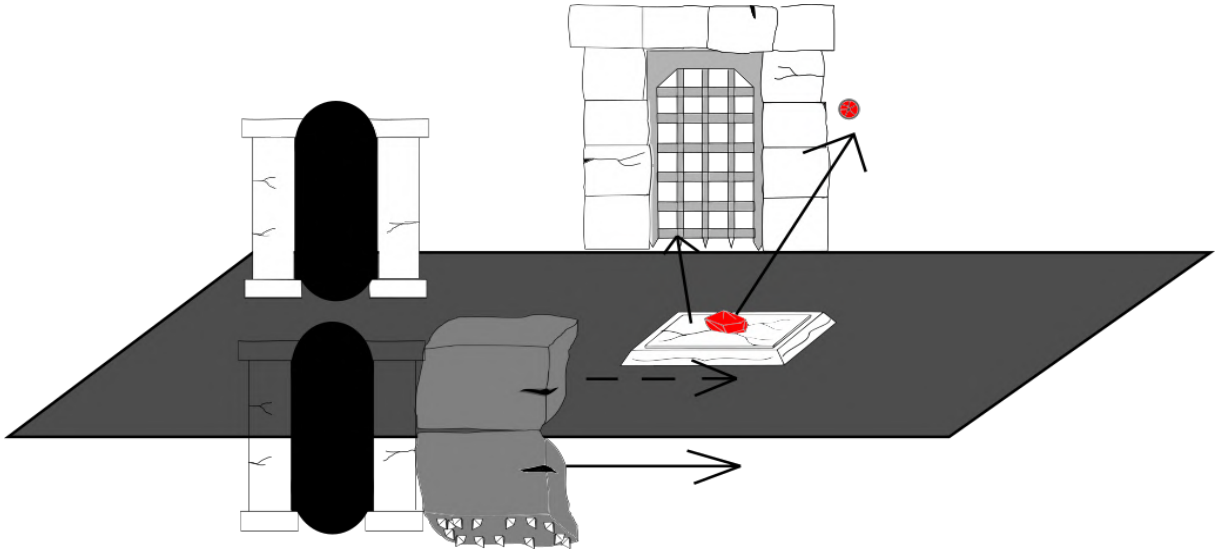


Figure 4.10: The first room of the tutorial level. The pressure plate opens the door. Notice the same color of the plate's crystal and the door's crystal.

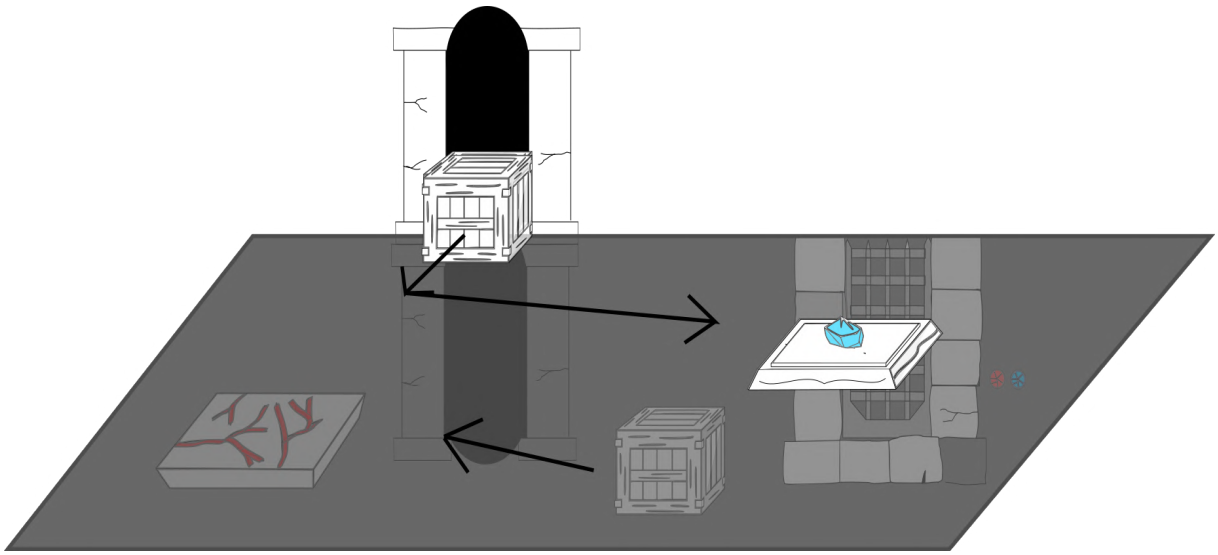


Figure 4.11: The second room of the tutorial level.

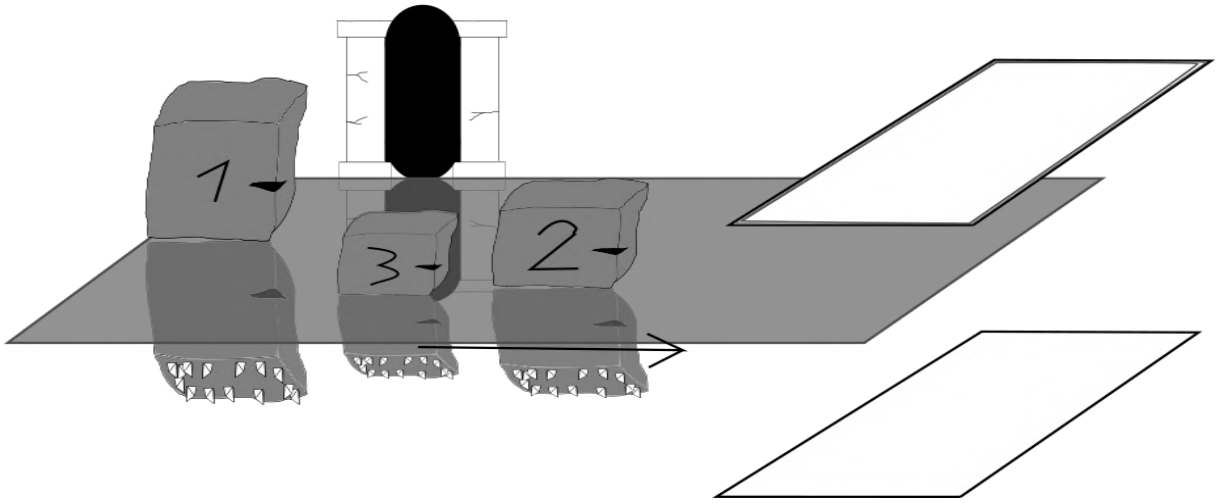


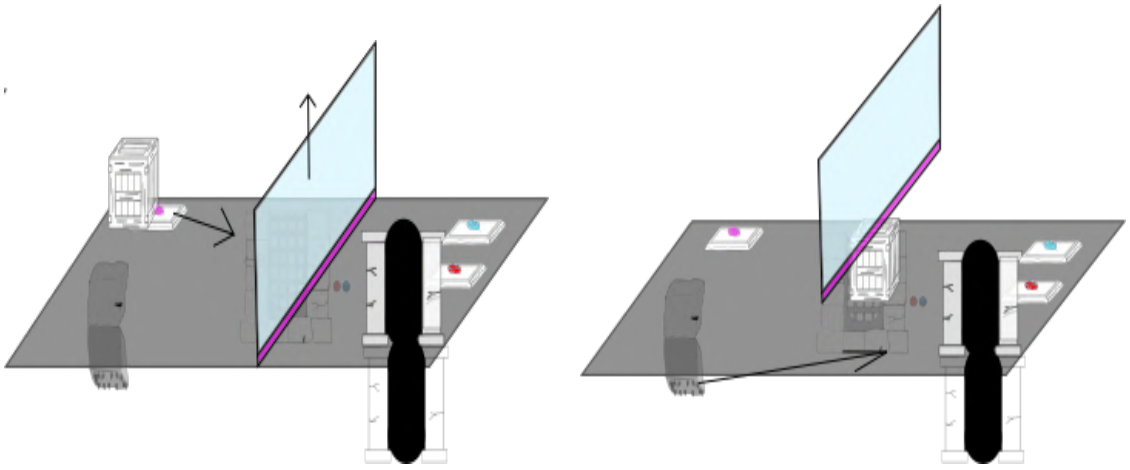
Figure 4.12: The seventh room of the tutorial level.

in the up world is significantly lower than the one in the parallel world. The task is to move the mirroring stones close to the platform in order according to their height (the numbers in Figure 4.12 represent the order, where number one is the highest and the first that needs to be moved). Finally, the player can jump on these stones in the up world and get on the platform.

The first level

The first level is a little more complicated than the tutorial level. There are more complex rooms that consist of more complicated combinations. There is not usually only one combination, but more of these combinations intertwine and connect.

The first level introduces other new elements. There is now also a button, a lift, and a glass barrier. The first level can be entered from the middle room by moving a crate on the pressure plate with red crystals. This closes the door to the tutorial level and opens the door to the first level.



(a) The barrier is opened by the pressure plate sharing the color of the barrier's bottom

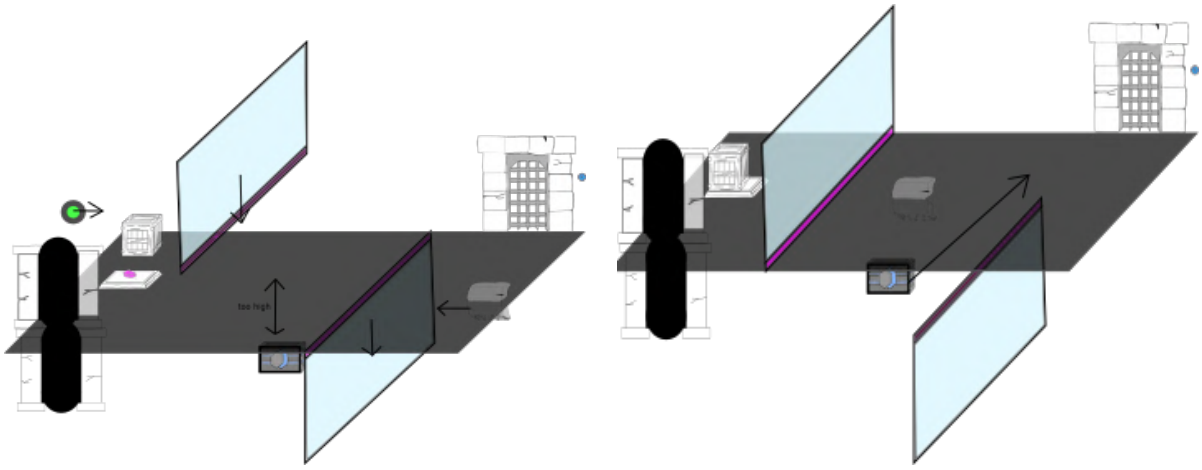
(b) The crate stopping the barrier from falling

Figure 4.13: The first room of the first level

The first room (Figure 4.13) shows the player the glass barrier and teaches them how it works. There is a barrier in the up world that is kept high by a pressed pressure plate. On the pressure plate, there is a crate. There are also two other pressure plates and the portal behind the open barrier. The mirroring stone is standing next to the crate. If the crate is moved from the plate, the barrier falls down. The player must move the crate next to the barrier and step on the plate herself/himself. After the barrier is fully up, the player can quickly move the crate under the barrier, stopping it from falling. Then s/he can change worlds and move the mirroring stone on one of the other plates. The crate needs to be pulled on the remaining pressure plate to open the door in the parallel world.

The third room (Figure 4.14) contains another new feature that is presented in the fourth room: a button. The button blooms with blue color as well as the only crystal placed next to the door in the up world. However, the button is situated in the parallel world. There are two glass barriers. One is in the up world and the other in the parallel world. The only pressure plate in the room controls both of these barriers. If the plate is pressed, the up world barrier falls down, and the parallel world barrier rises. On the other hand, when the plate is released, the up world barrier gets up, but the parallel world barrier falls. The plate is placed in the up world right next to a crate and the portal in front of those barriers. At the other end, next to the door, there is a mirroring stone behind the second barrier.

The problem here is that the button is too high for the player to reach. The right path is to move the crate on the plate, which switches the positions of both barriers. The up world barrier is currently down and the parallel world barrier is up. The player rotates the worlds and moves the mirroring stone under the button, enabling her/him to jump on it and press the button. The door opens. The player must now return to the up world, push the crate from the plate to clear the way, and leave.



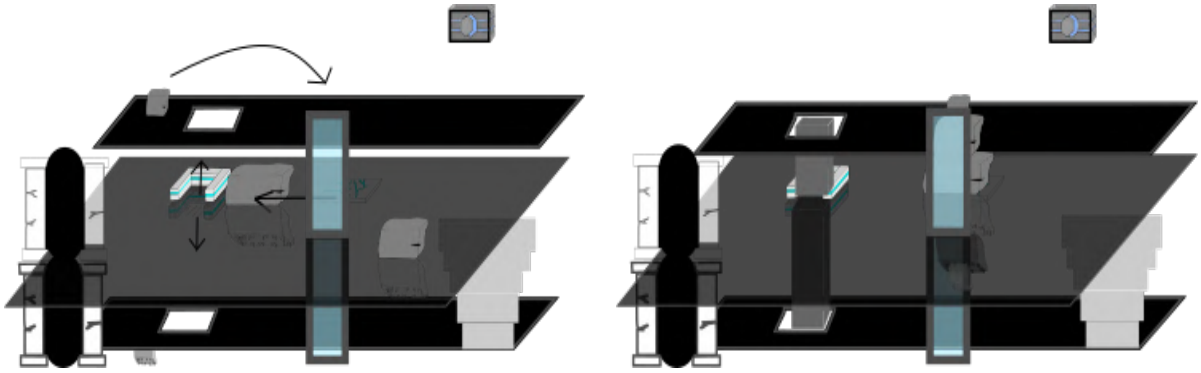
(a) The pressure plate is not pressed. The up world barrier is up, while the parallel world barrier is down. Green point signalizing where the player enters the room

(b) The pressure plate is pressed. The up world barrier is down, while the parallel world barrier is up

Figure 4.14: The third room of the first level. The raised barrier does not bloom, and its line has a darker tone of the pink color, while the closed barrier's line shines.

The seventh room (Figure 4.15) is parallel to the eight-room. This means that the player can choose which one s/he starts with. Other than that, there is only one exit, which means the player leaves using the same door as s/he enters. There is a platform in both worlds, but reachable by stairs only in the upside-down world. There is a mirroring lift that can lift the stones on the platform. The lift has already been presented in the fifth room. There are three mirroring stones and one pressure plate in the parallel world, which operates the lift. In the up world, there is a button controlling the door in the previous room. The button is reachable only

from the platform. To get onto the up world platform, the player needs to, using the lift, throw the stones down from the parallel world platform to build a tower. S/he can climb it to get to the button in the up world.



(a) The lift is not activated

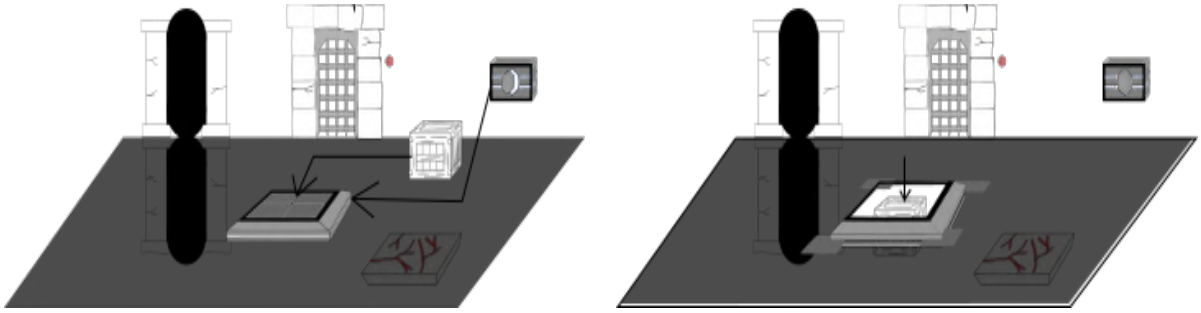
(b) The lift is activated and all the stones create a tower

Figure 4.15: The seventh room of the first level. The main task is to build a climbable tower of the three stones using the lift. The glass barrier is there to help to control the falling of the stones.

The second level

The second is the last level. In the last room of this level, there is the third crystal. After obtaining this crystal, an escaping portal in the Middle room is activated. After entering the portal, the game ends.

This level is constructed from ten much more difficult rooms and more improved rooms. It takes more time to solve these puzzles. At this level, a couple of new objects are introduced. The most important one is the teleporting plate, which creates most of the puzzles. Other new objects are a creating box and a moving platform.



(a) The teleporting plate is closed. Can be activated by a button.

(b) The teleporting plate is activated resulting in teleporting the crate through the floor to the other world

Figure 4.16: The first room of the second level. This room introduces the teleporting plate (in the center of the image).

The first room (Figure 4.16) introduces the teleporting plate. There is only a crate, a pressure plate, a button activating the teleporting plate, the portals, and the teleporting plate in the room. The player must move the crate onto the plate, switch worlds, press the button that activates the teleporting plate, and push the now teleported crate onto the pressure plate. The door opens.

The fourth room (Figure 4.17) introduces a new minor feature special only for this room, which is not anywhere else. This feature is about not backing out from the portal, only going

forward. There are two blocks in the front and at the back of each portal that close according to the direction from which the player entered the portal, as visualized and described more in detail in Figure 4.18. In the room, there are two crates in the up world, the mirroring stones, a destroyable mirroring barrier, one teleporting plate, a button activating the teleporting plate, and two pressure plates. The mirroring stones are behind the destroyable barrier. One pressure plate is in the up world inside a glass area that is reachable only by a portal. This pressure plate opens the door. The other one is in the parallel world and destroys the destroyable barrier.

This puzzle is solvable by using one of the crates to jump across the glass getting inside the glass area. However, at first, the player must teleport the second crate to the other world. Then, when jumping across the glass and into the portal, the player gets to the parallel world outside the glass area. Moves the teleported crate on the pressure plate, destroying the barrier. S/he must pull the mirroring stones outside to use them to push the crate in the up world between the pillars to get it inside the glass area. The player eventually teleports to the up world into the glass area, pushes the crate further onto the pressure plate to open the door. The player then uses the crate again to jump across the glass and leaves.

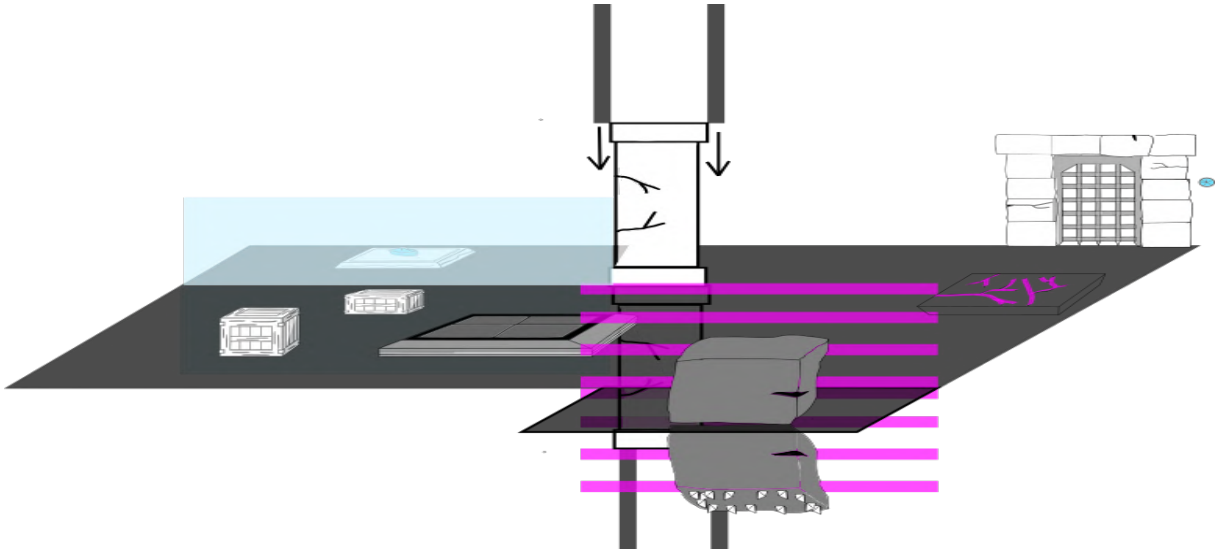
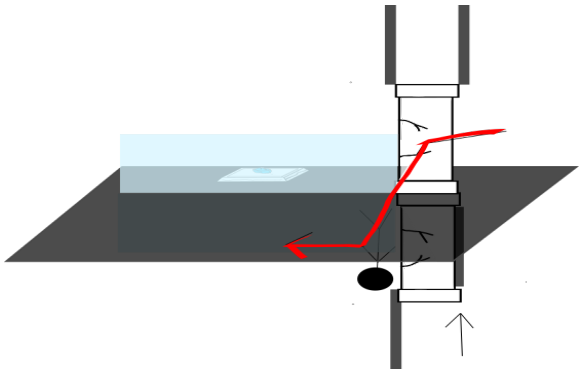
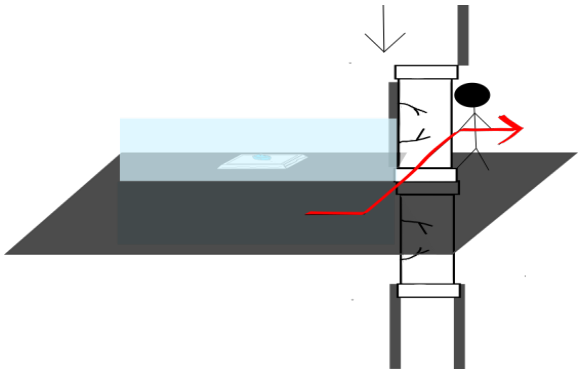


Figure 4.17: The fourth room of the second level. The main elements of this room are the teleporting plate (in the center), the glass area (on the left), and the closing blocks (above the portals)



(a) The player entered the portal from the outside area in the up world, resulting in the right parallel world closing block falling, disabling the player to go "backward"



(b) The player entered the portal from the glass area in the parallel world, resulting in the left up world closing block falling.

Figure 4.18: The closing block's functionality. The red line ilustrates the player's path

The fifth room (Figure 4.19) does not introduce anything new. There are two glass barriers in each world, a teleporting plate in the right bottom corner, and two mirroring buttons. One pair controls all barriers, and the other activates the teleporting plate. There is also a pressure plate in the up world, a crate in the parallel world, and two pairs of portals. The room is divided by the barriers into eights—the up world into quarters, as well as the parallel world. Looking at the room from the top, having the entrance door on the left and the exit door on the right, then the teleporting plate is in the left bottom quarter, the pressure plate is in the right bottom quarter with buttons activating the teleporting plate. The crate is in the top-left quarter. The buttons controlling the barriers are in the top-right quarter. The portals are in the right top and right bottom parts.

The player enters the room in the left bottom quarter and leaves in the right bottom one, all in the up world. The task is to use portals and switch barriers. The barriers may be in the same position in both worlds, but they are reversed. For example, if the up world horizontal barrier closes, the vertical barrier opens, but the parallel barriers behave inverted. The horizontal barrier opens, while the vertical barrier closes, as shown in Figure 4.20. The player must change worlds, switch barriers, which gets her/him to the crate, push the crate out of its quarter to the portal, again switch barriers, and continue pushing the crate to the other portals. The player must return to change barriers, which separates her/him now from the crate. For this reason, there are two portals. S/he must now change worlds and change again to get to the crate. The crate can now be pushed onto the teleporting plate. The player can activate the plate, which teleports the crate into the up world. Switching worlds get her/him also to the up world. However, the player is now again blocked by a barrier from the crate. Now it's just left to switch barriers for the last time, push the crate on the pressure plate, and the door opens.

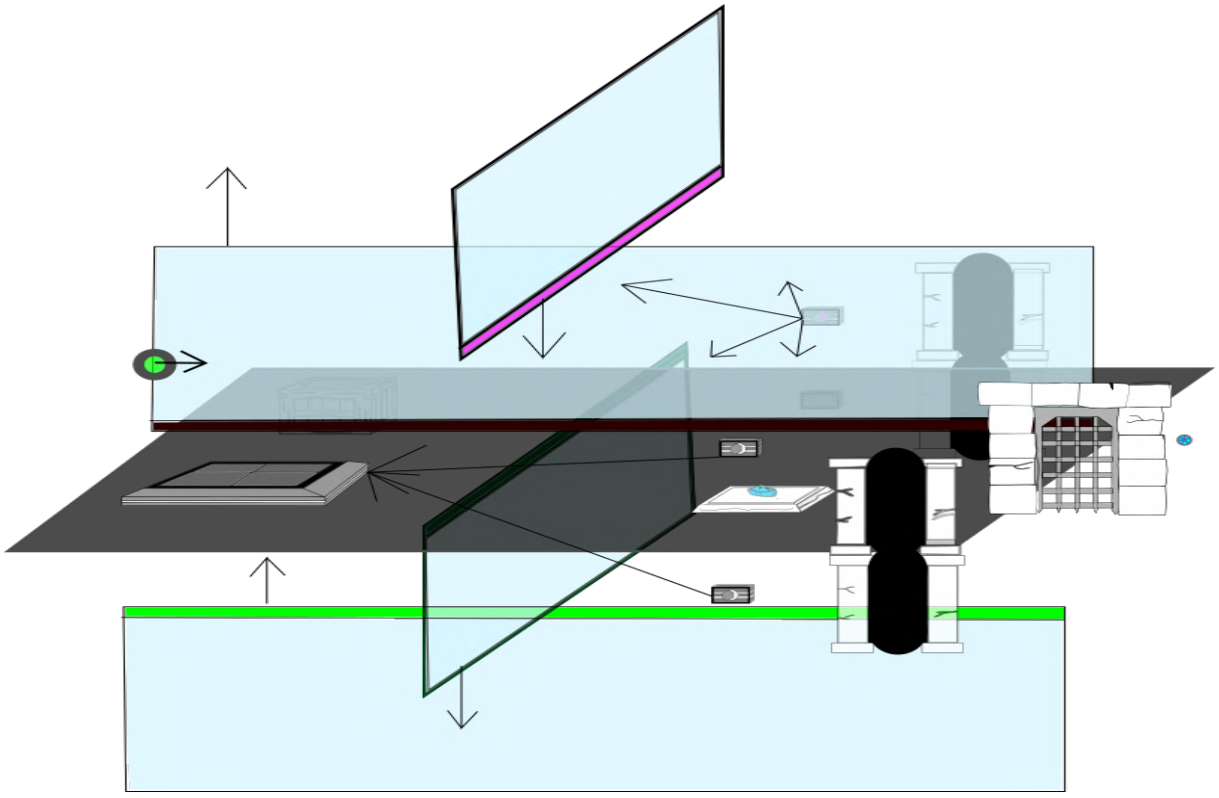


Figure 4.19: The fifth room of the second level. There are two pairs of barriers. Barriers in both worlds are inverted. Notice the different colors, preventing confusion with mirroring barriers. The room is divided by the barriers into eights. The green circle illustrates the place where the player enters the room.

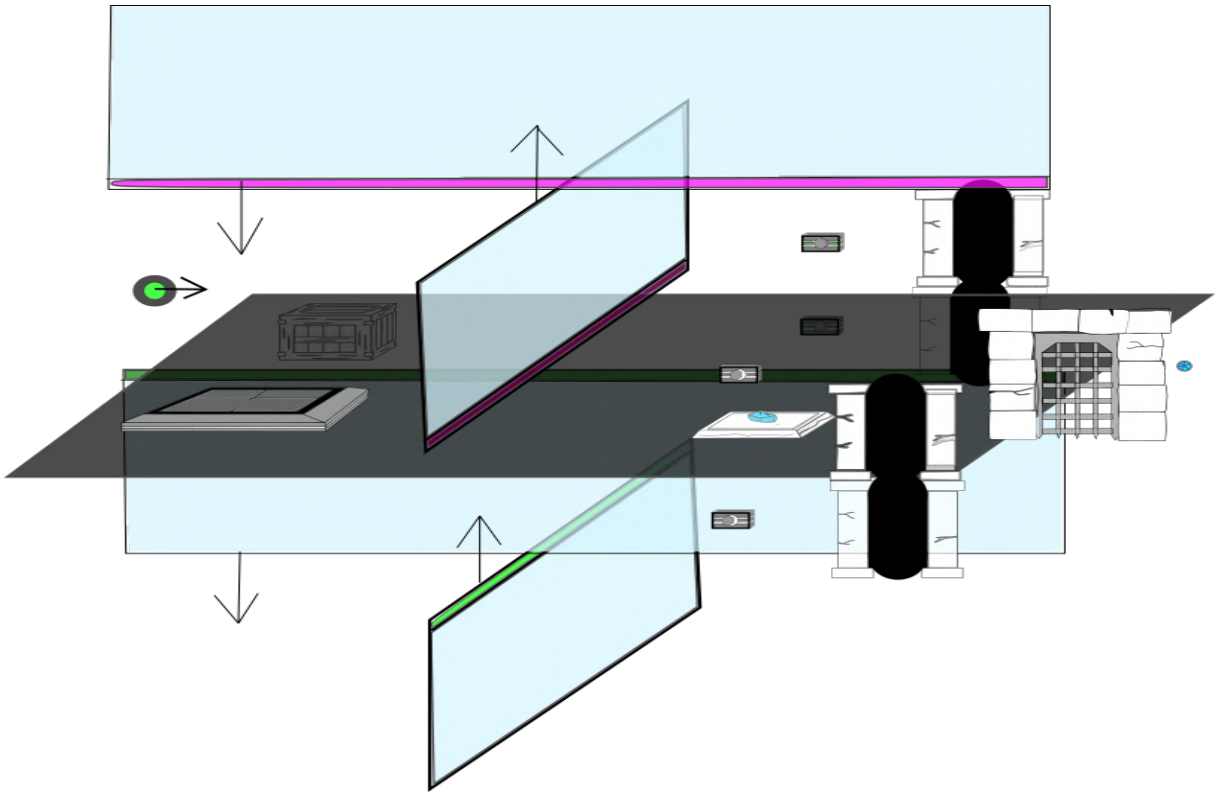


Figure 4.20: When the up world horizontal barrier opens, the up world vertical one closes. However, the parallel horizontal barrier closes and the parallel vertical opens and otherwise.

Chapter 5

Asset Creation Analysis

Modular components are beneficial during the process of level design. It can immensely simplify and speed up the overall work. To be able to create modular components, I have to look at what modular components are used for and why game developers resort to modular level design. Then, I need to analyze the creation of materials in the physically based illumination model and the process of texturing.

5.1 Modular Level Design

Modular components are being reused and combined to create various areas without the need of texturing and building every part of it on its own.

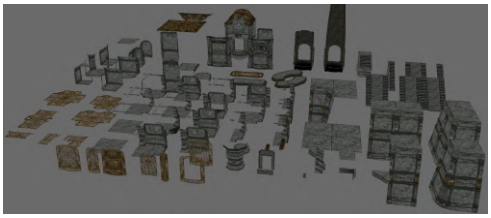


Figure 5.1: Modular components for skyrim [7]

The first thing that needs to be taken into consideration is the scale of modularity. There is a difference in how the player observes the scene. If the player sees the scene from a helicopter that flies above a capital city or if the game takes place in a house, where the player can observe from a short distance. In the first case, the components are the buildings or the entire city districts, whether in the second case, the assets are detailed parts within the house. [21, 14]

Then it is necessary to make a list of all components and key features that create the environment. Furthermore, the artist needs to create the geometry for specific occasions. For

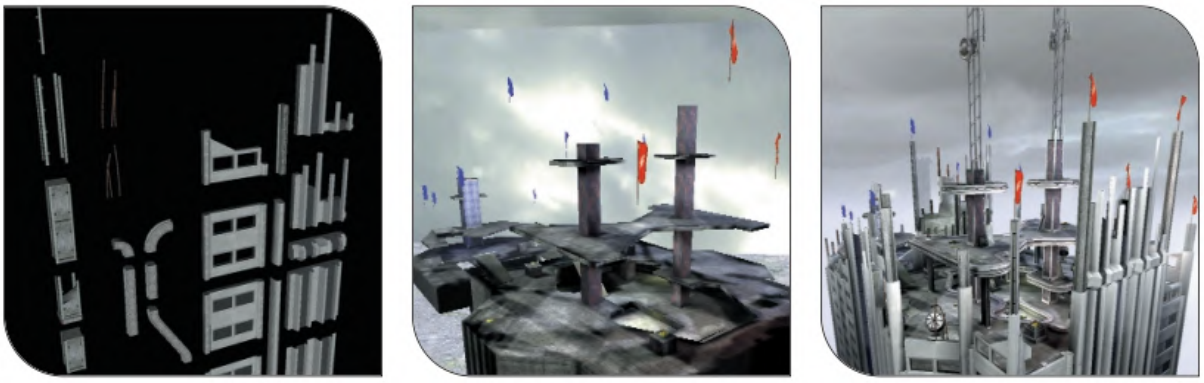


Figure 5.2: A small amount of modular components (left) can turn a basic structure (center) into a far more complex world (right) [21]

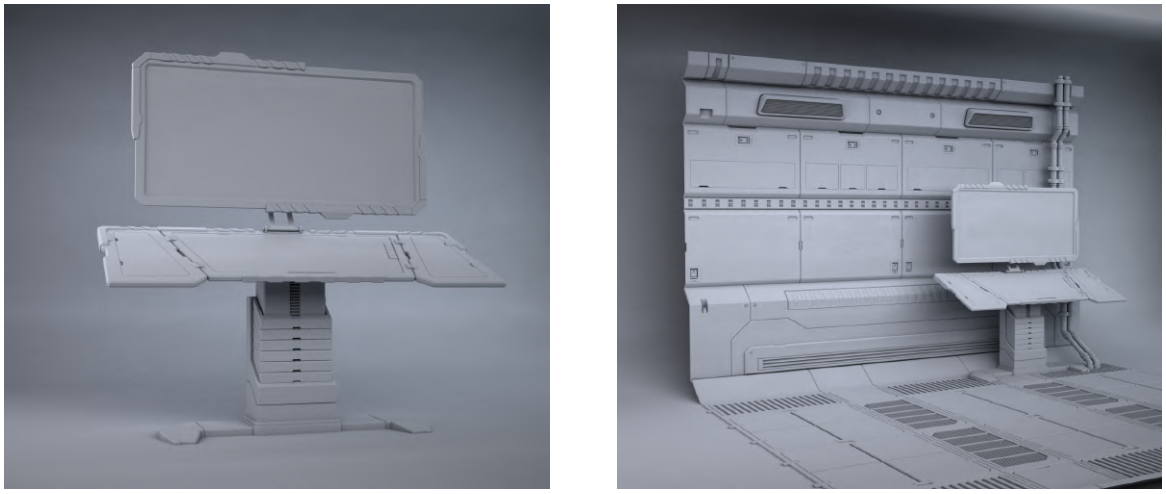
example, certain structures need capping off, and for that, creating a piece to cap off these structures. [21]

Starting with the basics is crucial in creating modular components. It is not advised to create detailed prefabs at first but to start with a base unit from which we can then create the variations. Assign more purposes to one asset. For example, a floor may also be a ceiling. Large structure does not have to be one single modular component, but variations of modular walls snapped together, allowing flexibility. [21, 14]

The next important thing is to accessorize the environment. Adding a few pieces that can disrupt the repetition improves the overall look, as shown in Figure 5.3. For example, two hallways created from the same modular pieces may look different by adding a statue or a few loose stones. A good idea is also creating accessories to hide levels of work. For example, when using a natural rock-looking texture on a wall, it may create seems in odd angles. The added accessories placed over the intersection may hide the seams in these parts, like a foliage or a large stone. [21, 14]

After having the basic models and enough accessories, the artist can create custom pieces. With these pieces, s/he can look if s/he can divide them into more parts that can be reused or if other sections can be added to these pieces to expand the area. And at last, is creating the level and snapping together all those modular pieces. [21]

Creating modular components has many benefits. It allows easier modifiability of the area. Suppose testers request another exit from a room with a modular level design, then it is no problem with this kind of approach. It also preserves consistency and saves memory. [21]



(a) The element **(b)** A complete module

Figure 5.3: An element used to disrupt the repetition of the environment [14]

5.2 Modeling Analysis

This section describes two techniques of 3D modeling useful for the creation of modular components.

3D modeling means creating a 3D representation of an object, more precisely a mathematical representation of an object's surface. It is then called a 3D model of an object. With a 3D model, we are able to capture size, shape, or even color of the object.

5.2.1 Modeling techniques

There are many ways how to create assets for games or movies. Choosing the suitable method is based on the project's requirements and what the artist or developer wants to approach.

I describe subdivision modeling and sculpting because these methods could be useful during modeling the modular components.

When using polygonal models in a game, balancing between how detailed the model is and the game performance is necessary. For this purpose, the artists use sculpting, which offers the artist easier modeling of details. The detailed model can be used for the baking of the normal map.

Sculpting

The sculpting technique, as the name suggests, simulates statue sculpting out of digitalized clay. An example of sculpting can be seen in Figure 5.4. Artists usually start with basic meshes. Using the right tools, like brushes, that enable sculpting techniques (e.g., pushing, pulling, pinching), the artist modifies the objects to fit their imagination. [13]

In this technique, artists work in layers, thus controlling the level of detail, similarly to the subdivision surface method. At first, the artist modifies the geometry to control the basic shape, subdivides the mesh, and adds more detail. [13]

With this method, it is easy to create small details, like scars, pores, and pimples, thus creating a realistic-looking model that may be needed in movies and games that use realistic visual effects. (In games, it is important to keep the least number of polygons as possible due to being computationally demanding when having a complicated mesh with a lot of polygons. Normal maps partially solve this problem).

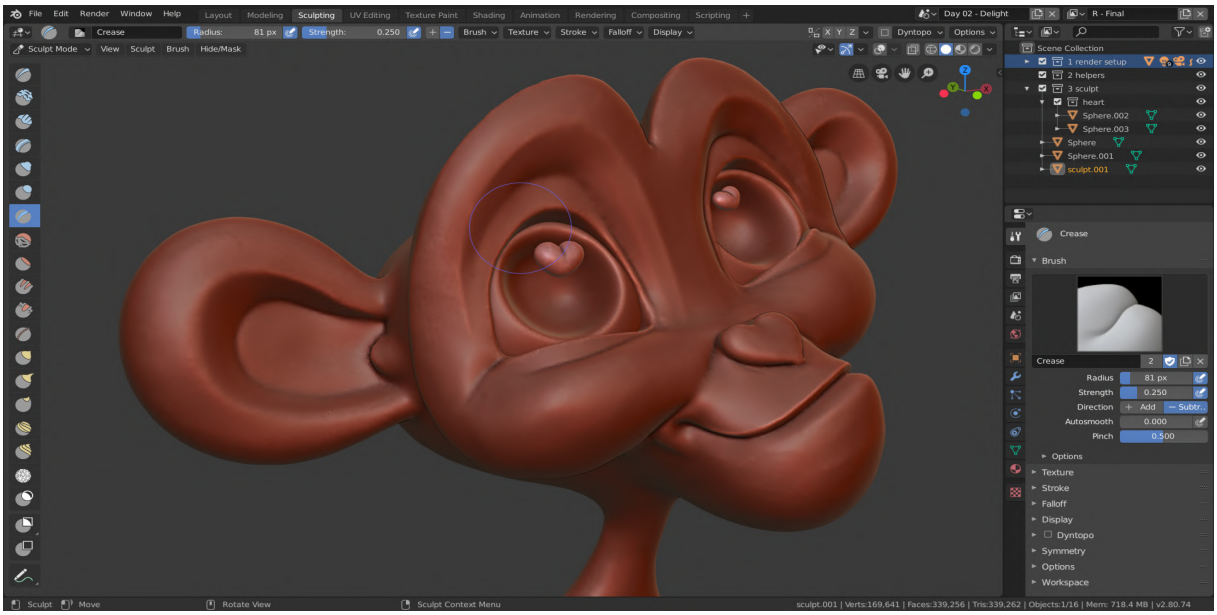


Figure 5.4: An example of sculpting in Blender [6]

Subdivision Modeling

Subdivision surfaces are surfaces that share the characteristics of polygons and NURBS[1] surfaces. They can create a smooth surface with only a few control points but still are able to control the local level of detail. They also allow sharp edges and the use of complex geometry

[1] Non-uniform rational basis spline

in complex regions of the mesh, as shown in Figure 5.5. Subdivision surfaces are continuous, which is good for animation because it preserves seams that can appear when animating NURBS surfaces. [3]

With subdivision surfaces, the artist can divide the surface until creating more detail and have better control over the area. For example, this is useful in modeling hands or just objects, where it is needed to have more detail. [3]

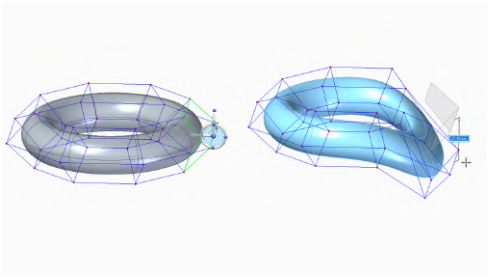


Figure 5.5: An example of subdivision surface modeling of an object of complex shape [19]

Subdivision modeling is sometimes referred to as box modeling because artists usually start with a basic shape, such as a cube (box), cylinder, and sphere, that they further modify to create the desired object. [23]

Modeling with subdivision surfaces is relatively easy because when the artist moves the control points to the desired position, it also reshapes the controlled area. Having the subdivided mesh at a lower level, moving controlling points reshapes a bigger area than at a higher level. [3]

This is heavily used in the method artists adapted when creating models with subdivision modeling. At first, they modify low-polygon meshes to change the overall shape, and then, they divide the mesh and create details on this more subdivided mesh. [1]

5.3 Materials Analysis

Materials define how light is reflected from an object, which is also closely related to an illumination model. The illumination model is used to calculate the intensity of light that is reflected from a surface. [4]. Illumination models can be divided into two groups: physically-based models and empirical models. [35]

Because Unity uses the physically-based illumination model, this section describes this model and the creation of materials in this model.

5.3.1 Physically-based illumination model

The physically-based illumination model, also referred to as physically based rendering (PBR), is a group of rendering techniques whose goal is to create the most similar lightning reflected as in the real world. The PBR models are more realistic than the empirical ones, and the materials are based on physical parameters, which means we get the same result regardless of the lightning. [20]

The PBR models are energy conserving, which means that every incoming ray is weighted to determine how much energy is reflected in the ray's direction. They are based on the microfacet theory and use a physically-based BRDF.

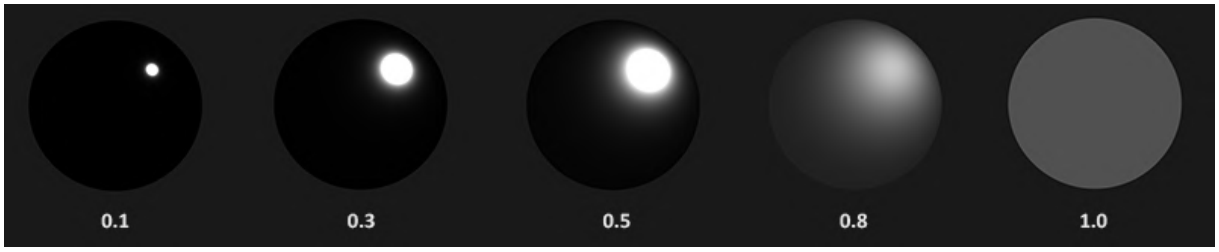


Figure 5.6: The specular reflection based in the roughness values

Microfacet theory says that every surface has tiny irregularities that reflect the light in different directions. These irregularities are called microfacets. The surface roughness defines how much these microfacets align. The more rough surface, the more widespread the specular reflection is because the rays are more likely to be scattered in different directions. In Figure 5.6 is shown that with a smaller roughness parameter, the specular reflection is much smaller and sharper and otherwise. [20]

The BRDF, the bidirectional reflective distribution function, describes how a ray is reflected from a surface. For a Cook-Torrance illumination model, the BRDF is calculated as

$$f_{Cook-Torrance} = \frac{DFG}{4 * (\vec{V} * \vec{N})(\vec{L} * \vec{N})} \tag{5.1}$$

,where $\vec{V}$ is the view direction, $\vec{N}$ the normal of the surface, and $\vec{L}$ is the light direction. D stands for a normal distribution function, G for the geometry function, and F for the Fresnel function.

The distribution and geometry functions are based on the microfacet theory. The distribution function defines the roughness of the surface, the geometry function describes the shadowing of the microfacets, and the Fresnel function tells how much specular light is reflected at which angle between the view and the normal vector. When the angle increases, the light also increases. Figure 5.7 shows the results of different Fresnel parameters.

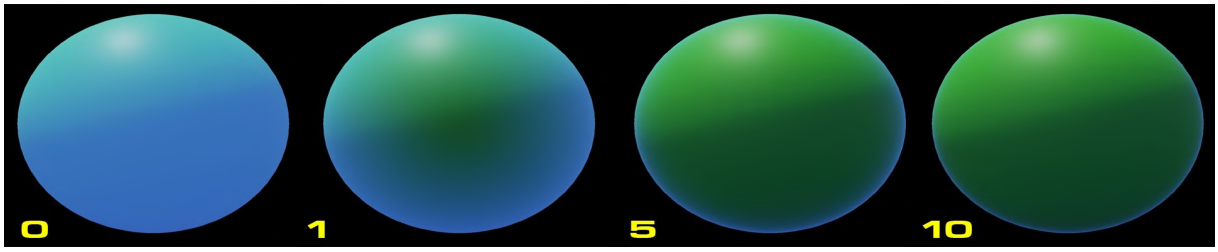


Figure 5.7: With higher values of the fresnel parameter, the fresnel effect is closer to the edge of the object [34]

5.3.2 Creating PBR materials

PBR materials are based on physical parameters on which depend how much light is reflected from the object. These parameters may be represented by textures. Usually, these parameters are: [20]

Albedo texture sets the color of the surface.

Normal map texture defines the normal vector of the point on the surface. It creates an illusion of a bent surface even though the surface is flat.

The metallic property says how much the surface behaves as a conductor or as an insulant.

Roughness specifies the roughness of the surface.

Ambient Occlusion defines the extra shadowing of the surface.

Unity instead of roughness, uses the smoothness parameter, which behaves in the same way as roughness but is inverted.

5.4 Textures Analysis

A texture is a function that returns the specific characteristics, such as color, metallicity of the surface, roughness, or the normal vector, for one point on a surface.

Using a texture may hugely improve the visual quality of an object at a relatively small cost. It is much better practice to create a texture that gives us almost the same result than

increasing the detail of the geometry mesh.
citemoderniPocitacovaGrafika

The essential part of using a texture on an object, apart from creating the texture, is texture mapping. Texture mapping is a process of assigning texture coordinates to every vertex of the 3D object. The texture coordinates for every point of a face are obtained by interpolating the texture coordinates of the face's vertices.

The process of translating the 3D geometry to the 2D space of the texture is called UV unwrapping. There are many methods of UV unwrapping, but the most common are: unwrapping with seams, view projection, unwrapping based on a limit angle, and box, or sphere, or cylindrical projection.

During unwrapping with seams, it is necessary to mark certain edges as seams. This cuts the geometry in these places and transforms it to the 2D space of texture.

Unwrapping based on a limit angle is more automatized unwrapping with seams, as the edges are automatically considered as seams according to the angle between the normal vectors of the face.

5.4.1 Creating textures

Textures can be created by painting, texture baking, or by texture cloning.

The texture during the process of texture painting, as the name suggests, is created by painting, as shown in Figure 6.5.

Texture cloning is mainly used when the artist has a real-world photograph and wants to use it on the model.

Texture baking renders the 3D object into the 2D texture. This is especially useful with normal maps that can be baked from a high poly model and used on the low poly model. We receive almost the same result, but with a lower number of faces. Apart from the normal maps, the current lightning of the 3D object can be baked into a texture.

Chapter 6

Creating Modular Components

In this chapter, I describe the process of the creation of modular components. I specify which modular components I detected to be needed to create the puzzle game presented in section 4 Game Design Document. Then, I describe the modeling of these components in Blender and preparing their materials and textures.

6.1 List of Identified Modular Components

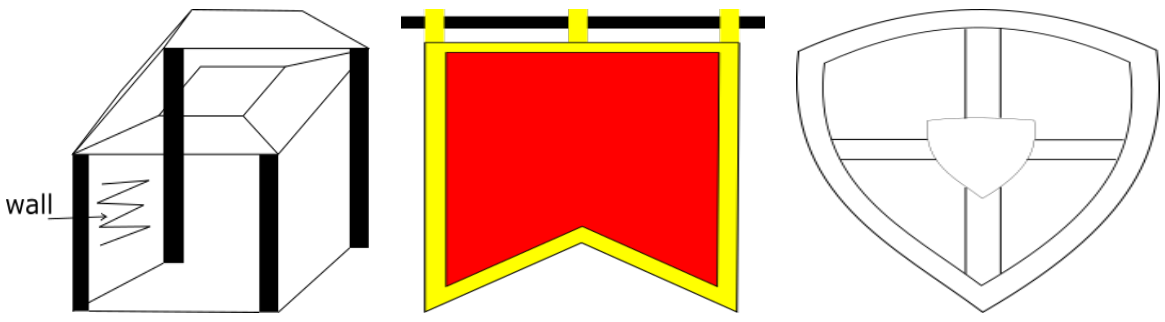
Based on the analysis presented in section 5.1 Modular Level Design of chapter 5 Asset Creation Analysis and in chapter 3 Game Design Analysis, specifically according to the section 4 Game Design Document, I identified the modular components needed for putting together a level in the game.

Because the game is set in a closed area, particularly in many rooms of a mansion, one of the most important modular components is the walls and a ceiling. The most basic wall is a plain block, from which there were derived different variations to meet the suggestion of having a few variations of one object that can be combined to disrupt the repetition. The wall types are: a plain wall block, a wall with a hole in the middle, a wall with a hole on a side, a wall with pillars, a wall with a fireplace, and a wall with cube decoration. Figure 6.2 shows the sketches of the wall's variations.

To easily combine the walls, the ceiling stands on four blocks that hide the walls' corners, as shown in Figure 6.1(a).

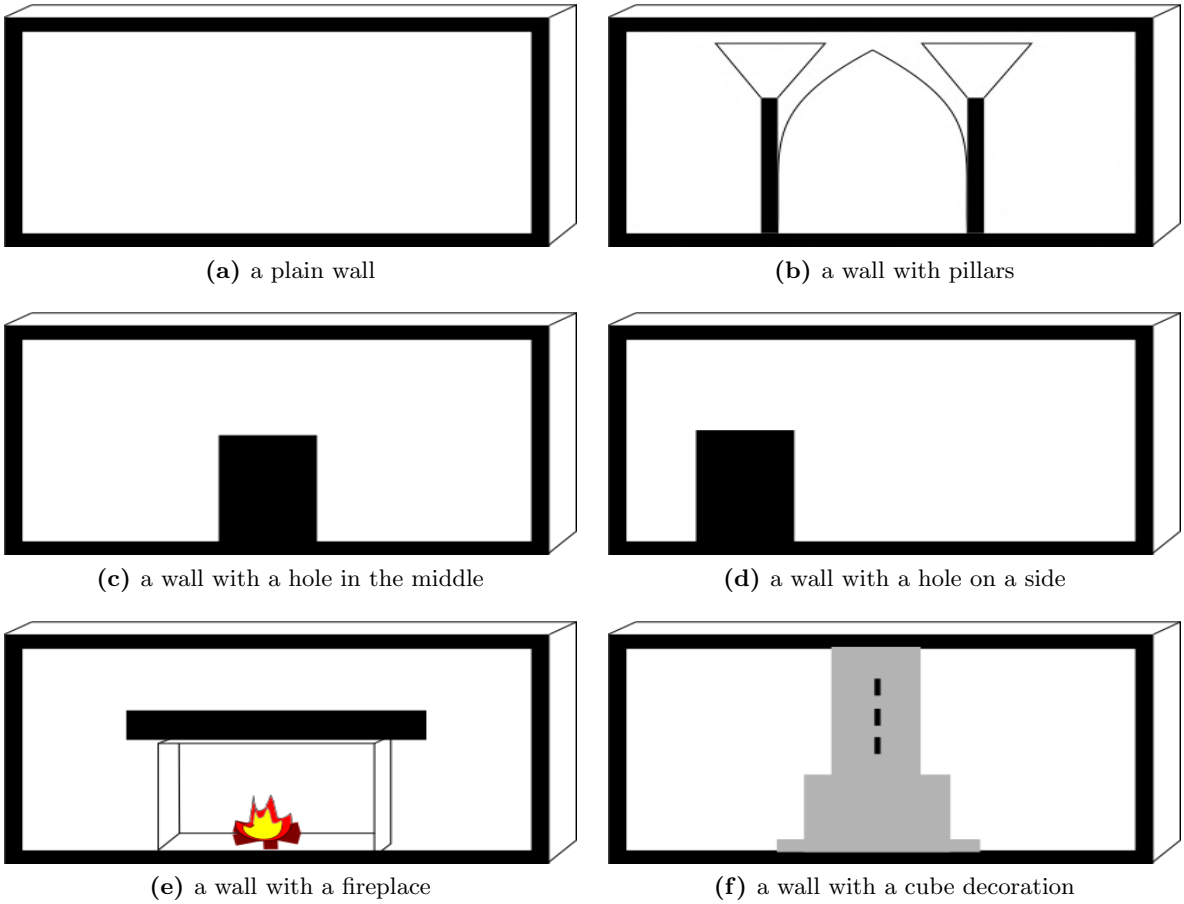
Modular components with no purpose but to accessorize the environment are the torch, the banner, and the crest, illustrated in Figure 6.1(b) and (c).

Other modular components were presented in the 4 Game Design Document section. These are the door, the lift's barrier, the portal, the teleporting plate, the pressure plate, the moving platform, the crystal, the stone, the crate, and the button.



(a) the ceiling **(b)** the banner **(c)** the crest

Figure 6.1: Modular components



(a) a plain wall

(b) a wall with pillars

(c) a wall with a hole in the middle

(d) a wall with a hole on a side

(e) a wall with a fireplace

(f) a wall with a cube decoration

Figure 6.2: Different wall variations

6.2 Modeling and Material Creation

I created polygonal models of the modular components using Blender. Then, I downloaded the textures and created materials for the models. The modular components were created in a way that they could be easily combined.

To create the models of the modular components, I mostly used Blender. All models were built based on the design presented in section 4 Game Design Document and section 6.1 List of Identified Modular Components using Blender's tools such as vertex, edge, and face transformation, extruding, knife tool, bevel, and various modifiers. An example of the use of modifiers during the creation of the crest model shows Figure 6.7. I downloaded textures from a website that offers CC0 textures [30], from which the material was constructed in Unity with the HDRP/Lit shader, as shown in Figure 6.8(d).

Certain materials were created only in Unity, such as the glass material, floor material, portal particle material, or bloom materials, shown in Figure 6.8(a), (b), (c).

Other materials and textures were created in Blender using the principle BSDF material modified in the Shader Graph, which can be seen in Figure 6.3. A few of the diffuse textures were painted using the Texture Paint tool. An example of painted textures can be seen in Figure 6.5 and Figure 6.6(a). I also created certain normal maps by creating a more detailed version of the object with the Multiresolution modifier and sculpting, which can be seen in Figure 6.4. The result was baked using the low-resolution and high-resolution version into a texture. I used this method to obtain the door's, the middle room door's, the portal's, and the crate's normal map texture, as shown in Figure 6.6(b).

In Figure 6.13 is shown an example of combining modular components to create a level.

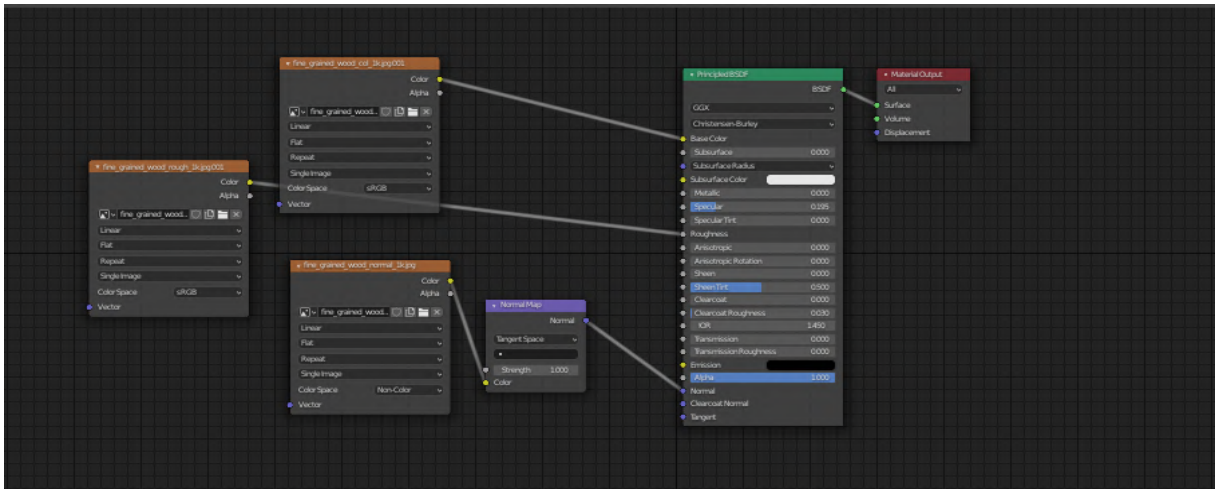
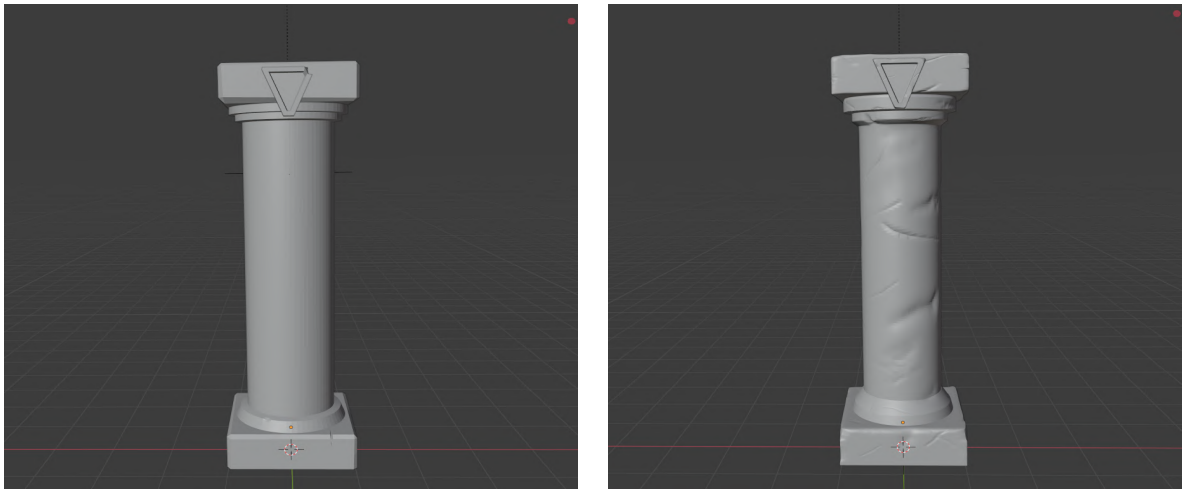


Figure 6.3: The shader graph created in Blender for a wooden part of the banner model



(a) The low-poly version of the pillar model **(b)** The high-poly version of the pillar model

Figure 6.4: Two versions of the pillar model used in the creation of the normal map

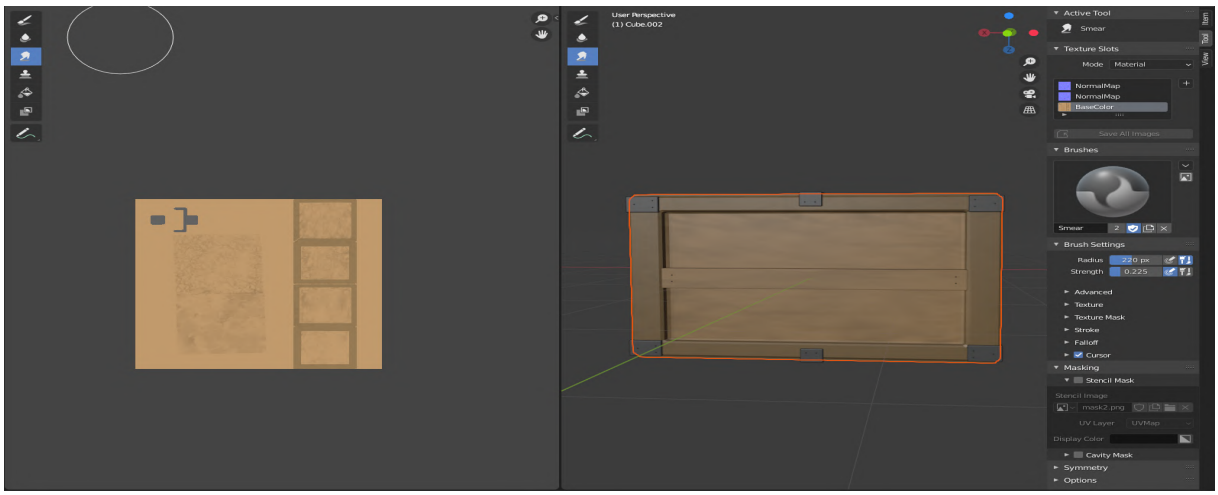
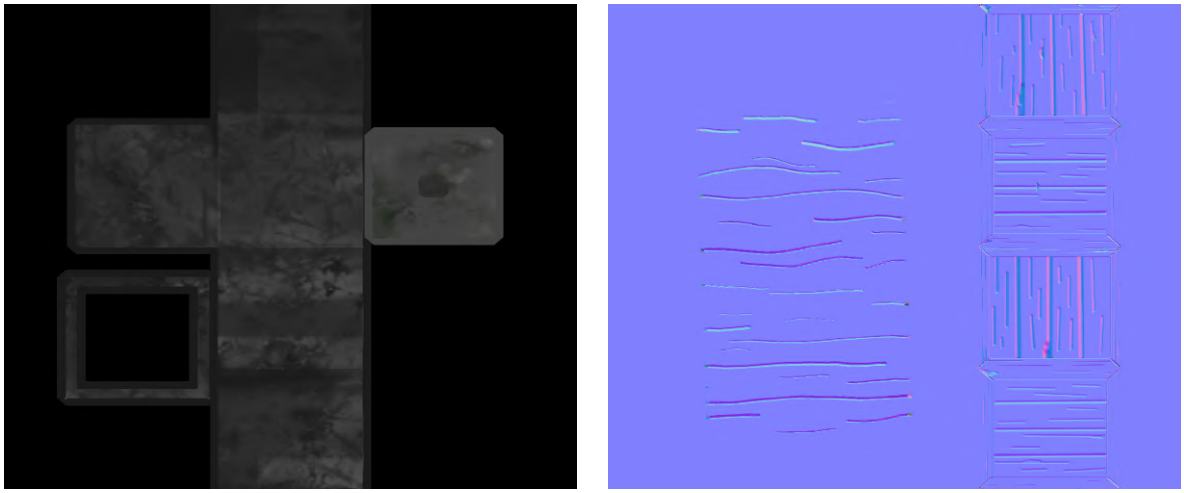


Figure 6.5: Painting a diffuse texture of the crate model



(a) The diffuse texture of the pressure plate model **(b)** The normal map texture of the crate model

Figure 6.6: Created textures in Blender

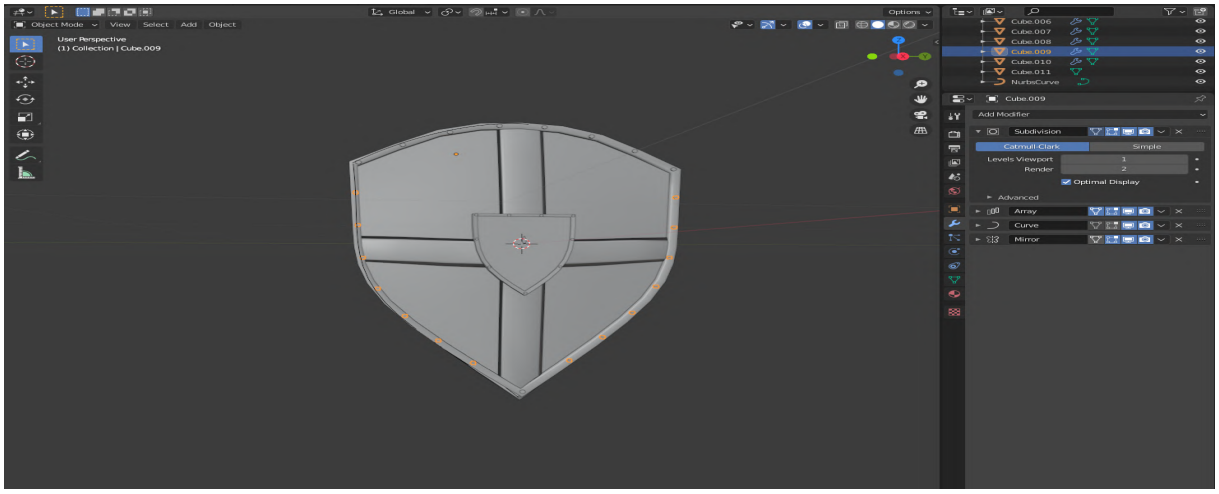
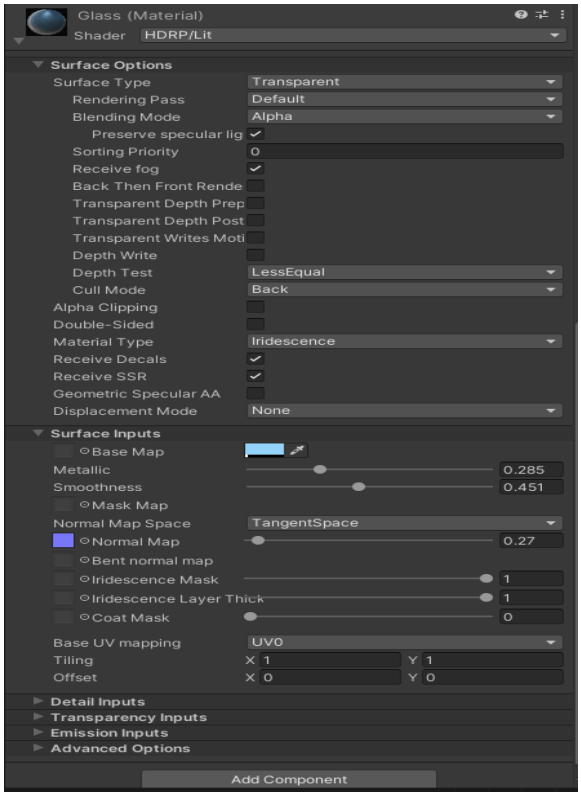
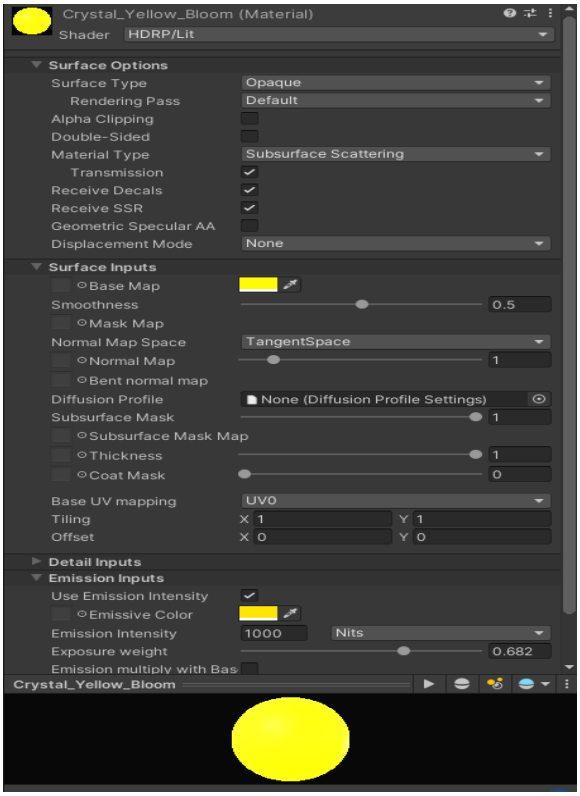


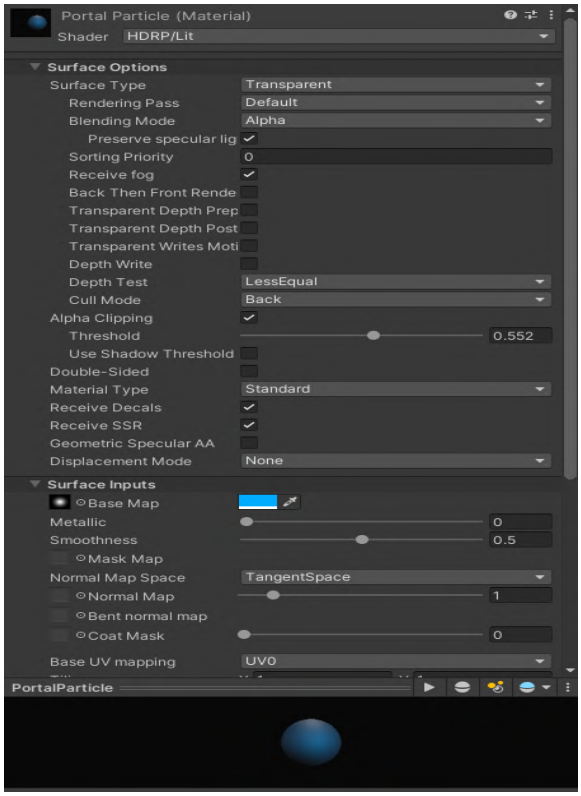
Figure 6.7: The modifier stack of the screws of the crest model



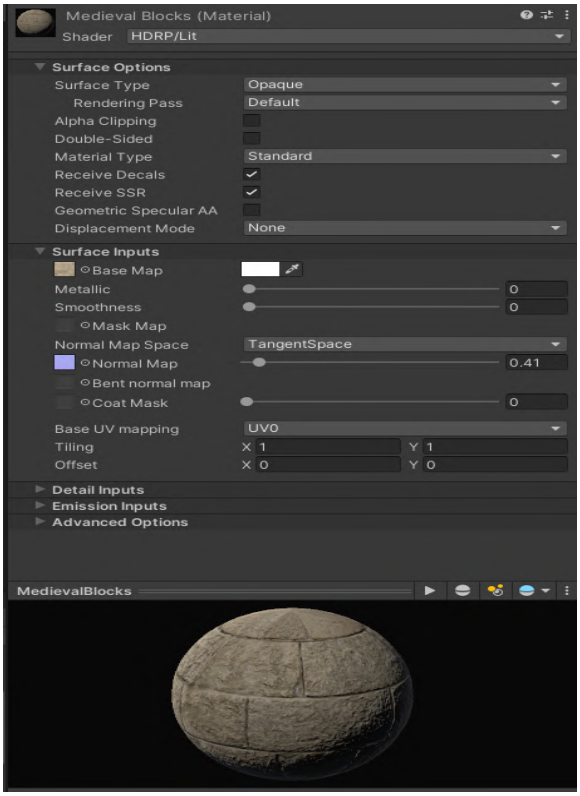
(a) The glass material



(b) The bloom material



(c) The portal particle material



(d) The medieval block material

Figure 6.8: Materials created in Unity. (a), (b), and (c) were created mainly in Unity. For material (d), the textures were downloaded from texturehaven.com [30]

6.3 Results

I imported the models into Unity, added them their material, and created a prefab for each modular component. To every prefab, I assigned its specific components and scripts to define its characteristics and behavior.

As already mentioned, there are three types of modular components.

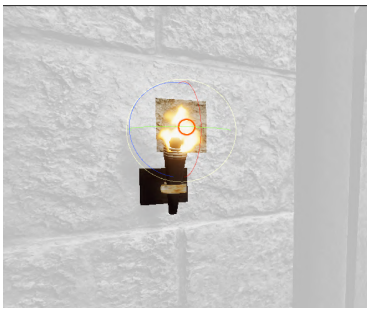
The first type is objects that must always be the same because of their purpose. These objects may differ only by the color of their crystal if they have any. Examples of these types of modular components can be seen in Figure 6.10.

The second type is models that have no purpose but to disrupt the repetition of the environment. These objects are randomly spread on the walls. In Figure 6.9, there are shown prefabs created from these components.

In Figure 6.11 are shown the wall variations, which is the last type of modular components. The walls were used to create rooms.

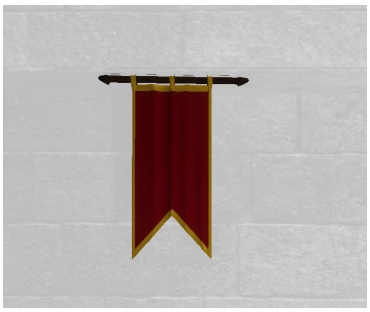
There are also components that appeared only once in the game. These include the main character, which is shown in Figure 6.12, or final portal in the Middle room, that shows already obtained crystals.

An example of combining these modular components to create the level is presented in Figure 6.13



(a) Torch

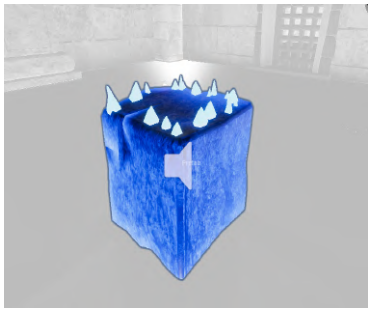
(b) Crest



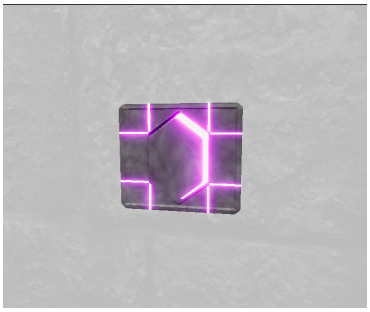
(c) Banner

Figure 6.9: Prefabs of components accessorizing the environment

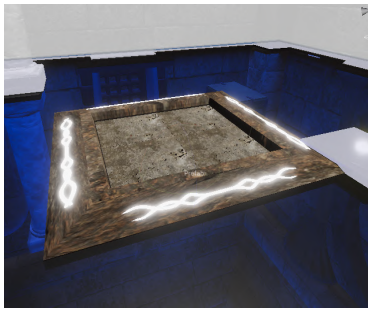
(a) Crate



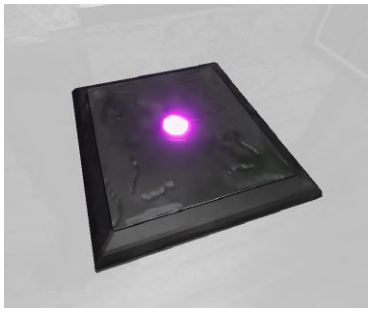
(b) Stone in the parallel world



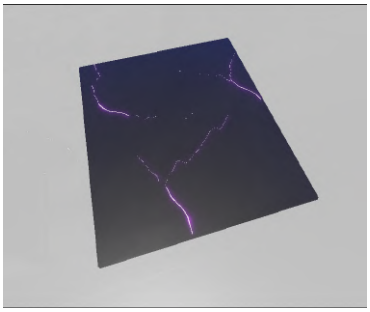
(c) Button



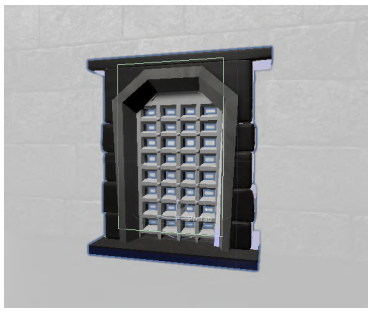
(d) Teleporting plate



(e) Pressure plate, from top



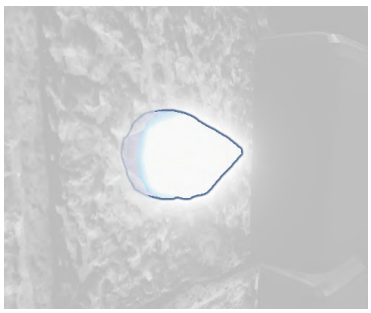
(f) Pressure plate, bottom part



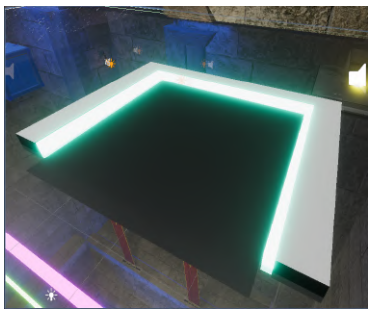
(g) Door

(h) Middle room's door

(i) The portal prefab

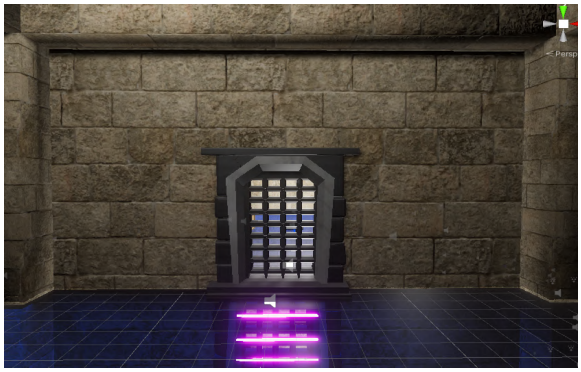


(j) Crystal

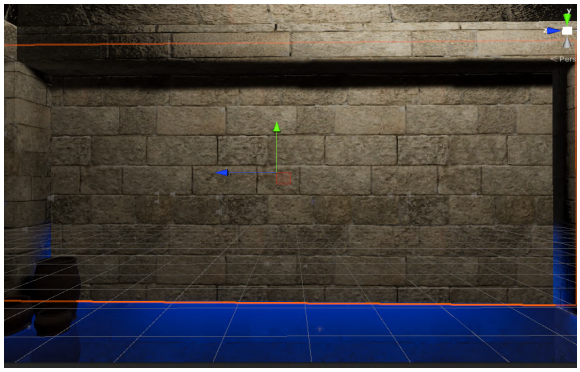


(k) Lift

Figure 6.10: Prefabs created from the modular components



(a) Wall with a hole in the center



(b) Plain wall

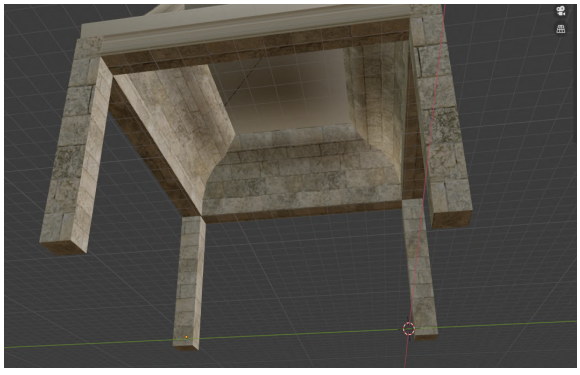
(c) Wall with pillars

(d) Wall with a fireplace

(e) Wall with a cube decoration

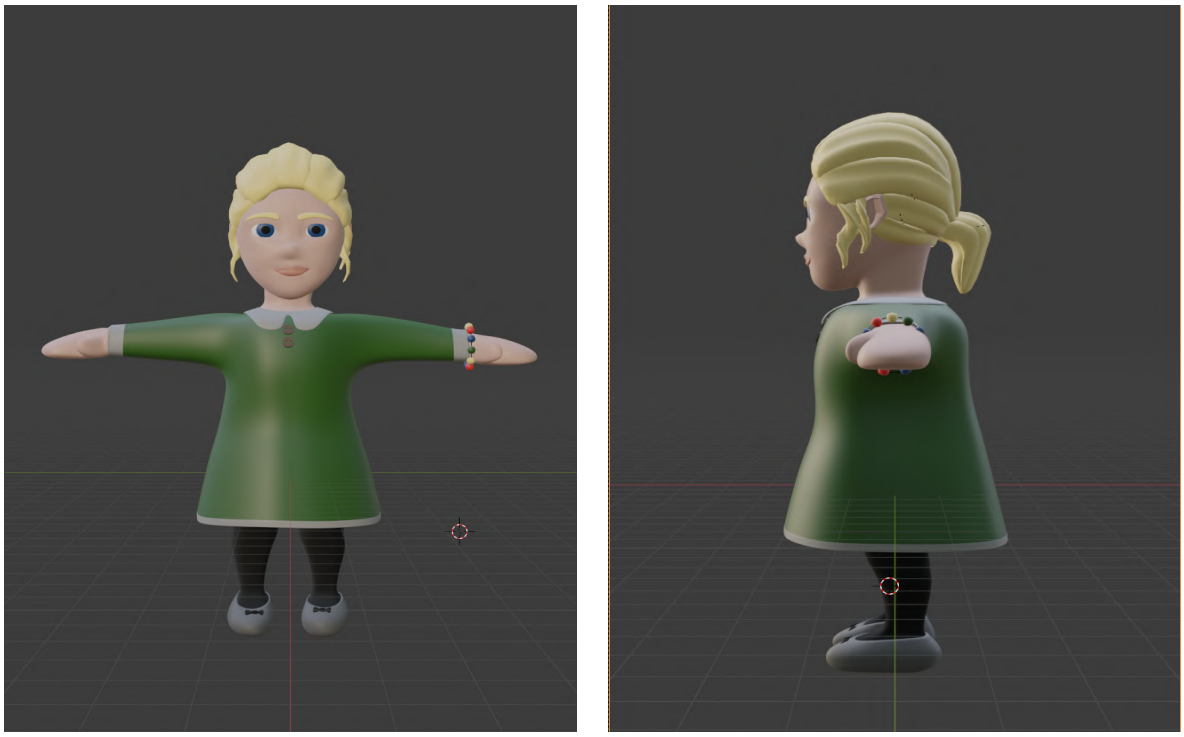
(f) Wall with a hole on a side

(g) Wall with a small hole



(h) Ceiling

Figure 6.11: All wall variations



(a) Front view **(b)** Side view

Figure 6.12: Different views of the main character

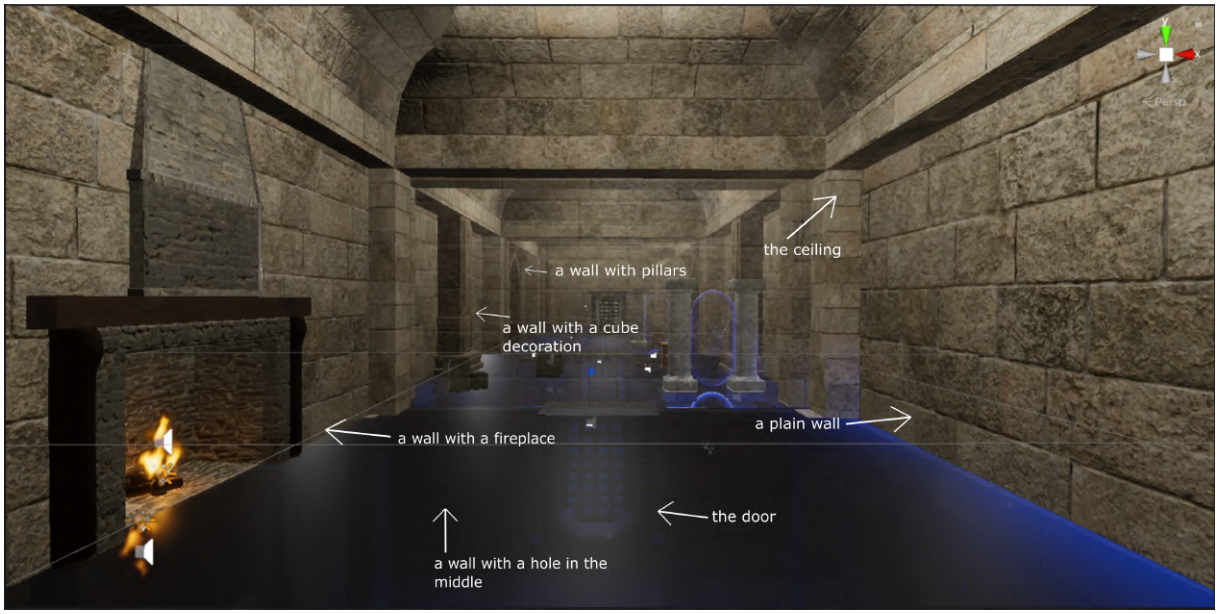


Figure 6.13: Level creation using modular components

Chapter 7

Implementation

In this chapter, I focus on the implementation of the game using the Unity engine, as presented in chapter 2 Unity Engine, based on the principles mentioned in 4 Game Design Document. I talk about every class and its utilization. After the class description, I show an example of one puzzle room created in the game using these classes and modular components presented in section 6.3.

7.1 Minimum Requirements

This section states the minimum requirements to play this game, which are: OS: Windows 10 64-bit, Processor: Intel Core i-5-4460 3.2 GHz, Memory: 8 GB RAM, Graphics: NVIDIA GeForce GTX 960

7.2 Used Technology

As mentioned, I created the game with the Unity engine, version 2020.1.10f. Scripts are written in C#, which is the language supported by Unity. For creating modular components, I used Blender [5] in version 2.90.1.

7.3 Project Description

The project consists of two scenes, the MainScene and the Menu Scene. Figure 7.1 shows the Menu Scene, which is the introductory scene that welcomes the player. There are three buttons that either start a new game, load the game from the last save, or quit. Choosing the two first options loads the Main Scene. This scene consists of three levels with 29 rooms in the up world and 29 rooms in the parallel world. These rooms were created according to the section 4 Game Design Document. This scene manages everything in the game as the game controls, the Camera, every game object, and its properties with assigned scripts. In Figure 7.3, you can see all rooms in the tutorial level, in Figure 7.4, there are all rooms in the first level, including the Middle room, and in Figure 7.5, you can see all rooms in the second level.

It also consists of every created model in the Assets/Models folder, used materials in the Assets/Materials folder, and textures saved in Assets/Materials/Textures. In the Assets/Prefab folder, there are prefabs created from the models already with scripts and components assigned ready to be used in the scene. In the Assets/Animation folder are collected all Animator Controllers and Animation Clips used in the game. There is also the Resources folder, which also contains the Animation folder and Video folder. When the game needs to load something from the script, it looks in the Resource folder. The Assets/Resources/Video folder contains all videos used in the tutorial. The Assets folder also contains the Sound folder, where all sound

effects are stored, the Scripts folder, where all implementation is held, and the Shader folder containing two shaders.

The project is an HDRP project, which means the scene is rendered using Unity's High Definition Rendering Pipeline. This introduces more realistic materials and lighting, and makes it easier to create shaders as well as particle systems and blooming effects.

Figure 7.1: The Menu Scene

7.4 Classes Description

In this part, I talk about each class and shortly describe what it does. As mentioned, the game loop is already implemented within Unity as basic physics and collider management, so this was not needed to implement.

7.4.1 Classes implementing movement

As mentioned in the Movement section of chapter 2 Unity Engine, the movement is partly implemented within Unity in Rigidbody and Character Controller components. It is necessary for understanding to mention that I use the Character Controller component only for the player and Rigidbody component for all other moving GameObjects.

The MouseView class rotates the player based on the mouse movement.

The PlayerMovement class manages the keyboard input and uses the CharacterController component's method *Move* to move the object in a given direction. It takes a Vector3 value of motion and returns CollisionFlags, which indicates the direction of a collision. The class also implements the gravitational force acting on the player and jumping if the player is not grounded when the spacebar is pressed. This is implemented in *FixedUpdate* which is mainly used for physics updates. The Update function checks if the left control key was pressed/released for setting the crouching and calling the right method that changes the player's size to simulate crouching.

The ColliderController class uses the *OnControllerColliderHit* method that implements pushing objects by the player. It uses the *AddForce* method of Rigidbody component that takes Vector3 value of velocity, which is calculated as the multiplication of the direction of

the push by force. It also checks if the pushed object mirrors to the parallel world and calls *UpdateMirObjectPosition* of **the MirroringObjectInfo** class that takes the calculated velocity.

The UpdateMirObjectPosition method of **MirroringObjectInfo** class calls within its body the *AddForce* method of Rigidbody component with the input velocity. It returns true if the position was changed. Otherwise, it returns false. *The LateUpdate* method checks if its local position is the same as the local position of its mirroring object. If it is not, it compares the velocity value of the Rigidbody component in the z and x coordinates of both objects. Then it sets the object's local position with a higher value in one of the coordinates of the velocity variable to be the same as the object's local position with the lower value. It is used if one of those objects is slowed down by an obstacle to preserve their mirroring.

7.4.2 Classes implementing the changing of the worlds

The RotatingWorlds class is assigned to the plane of the Portal prefab. Its *OnTriggerEnter* method of Collider component is called if the component touches another Collider component. The method checks if the other Collider component is assigned to a player by comparing its tag to the player tag. It sets the *playerIsOverlappig* variable to true. *OnTriggerExit* method then sets this variable to false. The Update method checks the value of the *playerIsOverlapping* variable. When true and if not already rotating, it starts the rotation of the worlds. Otherwise, it checks if the rotation has reached the desired angle and stops the rotation. It rotates all objects around the Camera so that the player does not move during the rotation. It uses the *RotateAround* function of the Transform component. It takes the center of the rotation in Vector3 format, the direction of the rotation, and rotation angle. The angle is calculated as the multiplication of Time.deltaTime and a rotation angle per second stored in the *rotAnglePerSecond* variable. It also calls the *StartRotation* method of **the CameraRotating** class.

The CameraRotating class manages the updating of the objects during the rotation by calling proper methods. At the start of the rotation, it calls *the UpdateRigidbody* method, *the UpdateMirroringObejcts* method, and *the UpdatePressurePlates* method of **the GravityObjects** class assigned to parents of all rooms in the UpWorld and the ParallelWorld. It also changes the player's object parent to be the object that does not rotate to avoid the player's rotation. At the end of the rotation, it again calls those three methods and sets the player's parent back. It updates rotated and *inParallelWorld* variables of **the GameManager** class.

The GravityObjects class updates Rigidbody objects, pressure plates, and mirroring objects when called to prevent them from moving during the rotation and falling when they are upside down. *The UpdateRigidbody* method at the start of the rotation sets all Rigidbodies to be kinematic. At the end of the rotation, based on whether the children's objects are currently upside down or otherwise sets *the isKinematic* property and *the useGravity* property of the Rigidbody component to true or false. *The UpdatePressurePlate* method disables BoxCollider and the PressurePlate components at the start of the rotation and enables them at the end. *The UpdateMirroringObjects* method enables and disables the **MirroingObjectInfo** component based on the stage of rotation.

7.4.3 Classes operating objects

There are three types of classes. Classes which only update their GameObjects, for example, **PressurePlate** class. Then there are classes, which operate other GameObjects, such as **PressurePlateFunc** class and finally classes, which are operated by other GameObjects, such as **OpeningClosingObj**.

All classes of the third type are created by extending an abstract class **ObjectOperation**, which defines *StartFunctionality*, *EndFunctionality* and *EndFuncUsingRigidbody* methods. These methods are then called by classes that control other GameObjects without having to distinguish which object they specifically control.

PressurePlate prefab's classes

The PressurePlate class changes the position's y value up or down of the plate if something steps on it or otherwise.

The PressurePlateFunc class calls within *the OnTriggerEnter* and *the OnTriggerExit* methods the *StartFunctionality*, *EndFunctionality* and *EndFuncUsingRigidbody* methods of the object stored in ControlledObj variable.

The PressurePlateFuncMirror class extends **The PressurePlateFunc** class. It has the same functionality as PressurePlateFunc class, but it is used on the mirroring plate. This plate can be triggered by the player when the other does not get triggered. However, when both get triggered by the mirroring stone, it would get to a loop if both plates had the same functions.

Button prefab's classes

The Button class controls the animation updating the button's middle part's position by moving it smoothly backward/forward.

The ButtonFunc class checks if the player clicks on the button and stands inside a specified distance. It calls the *StartAnim()/EndAnim()* function of the Button class to start the starting or the ending animation. It also manages the button's controlled object. It also turns off/on the blooming of the button's and the object's crystal by calling the *TurnOn()/TurnOff()* functions of **the BloomOnOff** class.

Door prefab's classes

The OpeningClosingObj and **the MainDoorOpening** classes update the position of the object when opened or closed in methods *GoUp* and *GoDown* using coroutines *SmoothlyGoUp* and *SmoothlyGoDown* that do the opening or closing continuous. *The GoDown* method also checks if the door has a Rigidbody component assigned to it. If it does, it calls *the GoDown* method of the **OpeningClosingRigidbody** class. **the MainDoorOpening** is extended from **The OpeningClosingObj** class, which is extended from **ObjectOperation** class.

The OpeningClosingRigidbody class uses the Rigidbody component to close the object. In that case, if there is an object under the door that would prevent the door from closing, the Rigidbody component makes the door fall on the object and not fall through it.

The ChangeRoom class within *the OnTriggerEnter()* function calls *the SetRoom()* function of **the GameManager** to set the player's current position used during saving. When the player exits the collider, the *OnTriggerExit()* function gets called, deactivating the game object.

Lift prefab's classes

The Lift class, extended from **the OpeningClosingObj**, and **the LiftRigidbody** class work the same as **the OpeningClosingObj** and **the OpeningClosingRigidbody** classes but for a Lift object. However, **The Lift** class does not update only the y coordinate of the object's position, but it also changes the y coordinate of the object's scale to create the effect of growing/shrinking.

TeleportingPlatform prefab's classes

The Teleporter class is responsible for throwing an object into the other world, switching its parent, updating its variables, while **the TeleporterOpening** class opens the platform by smoothly moving its four middle blocks along diagonals in opposite directions.

Barrier classes

The glass barrier to open/close itself uses the same classes as doors. However, the destroyable barrier uses **the BarrierDestroy** class, which activates/deactivates the barrier when called.

7.4.4 Other classes

GameManager manages the current game state. It pauses the game when the menu is active, quits the game, loads, and saves the game. **GameManager** contains many important variables controlling the game, such as *curNumOfCrystals*, *currentLevelsActive*, *currentRoom*, *isParallelWorld*. It also changes levels and activates/deactivates rooms according to the currently active level.

CreateBox using the stone's prefab creates its instance at a specified position inside the box.

The FinalCrystal class using *the OnTriggerEnter* function checks if an object entered its collider. The crystal then deactivates and activates the specific crystal in the Middle Room. It also increases the number of currently obtained crystals stored in **the GameManager** class.

The FootSteps class plays the footstep sound according to the main character's animation played during walking.

The BloomOnOff class aims to turn on/off the bloom effect on the object by switching the bloom material and one darker material that does not transmit light.

The MenuButton class is only used in *the MenuScene* to control the animation of pressing or releasing buttons by smoothly updating its position.

The AudioController class stops the audio playing of a movable object when the object stops moving.

The SlidingPlatform movement is mainly controlled by its rigidbody. But when it should move, how far and how fast is managed by its *SlidePlatform()* and *ReturnPlatform()* methods.

The GravityChange class simulates gravity in the opposite direction to every GameObejct at the start in the UpWorld with a Rigidbody component affected by gravity. It calls *the AddForce* method of the Rigidbody component and passes it a Vector3 with zeros in x and z coordinates and 9.18 in y coordinate.

Dialogue classes

The DialogueManager class, **the DialogueTrigger** class, and **the ObjDialogue** class control all the dialogue. **The DialogueManager** class has the biggest function by displaying and writing the sentences. It also plays a video if there is any to be played. It puts sentences in a queue by calling *the AddSentence()* function of **the ObjDialogue** class, which controls which sentences are added to the queue. *The StartDialogue()* function of **the DialogueManager** class is called by the *TriggerDialogue()* function of **the DialogueTrigger** class.

Saving system's classes

Saving system is controlled mainly from **the SaveSystem** class, which saves data or loads data. The data is saved using the **ObjectData** object. The created ObjectData object is filled with the crucial data by calling *the SetVariables()* of **the ObjectData** class before saving. During loading, the data is loaded into an ObjectData object.

7.4.5 Used additional assets

From Unity's Asset Store [32], I downloaded the AllSkyFree package containing multiple skyboxes. I used, in particular, the DeepDusk sky material.

To animate and rig the main character, I used a web page called Mixamo [18]. After uploading the model of the character, Mixamo automatically creates the rigged human skeleton that fits the uploaded model, enabling it to animation. Mixamo offers a database of various full-body animations that can be downloaded for free and used in Unity.

Sound effects were downloaded from a web page with a database of free sound effects [28]. Specifically, the credit for the main sound used in the game goes to Shade Davy [9].

I found the font used in the game on a page with many FFC fonts provided for downloading. I chose the Grange font [17].

The fire shader was created according to a video tutorial [24].

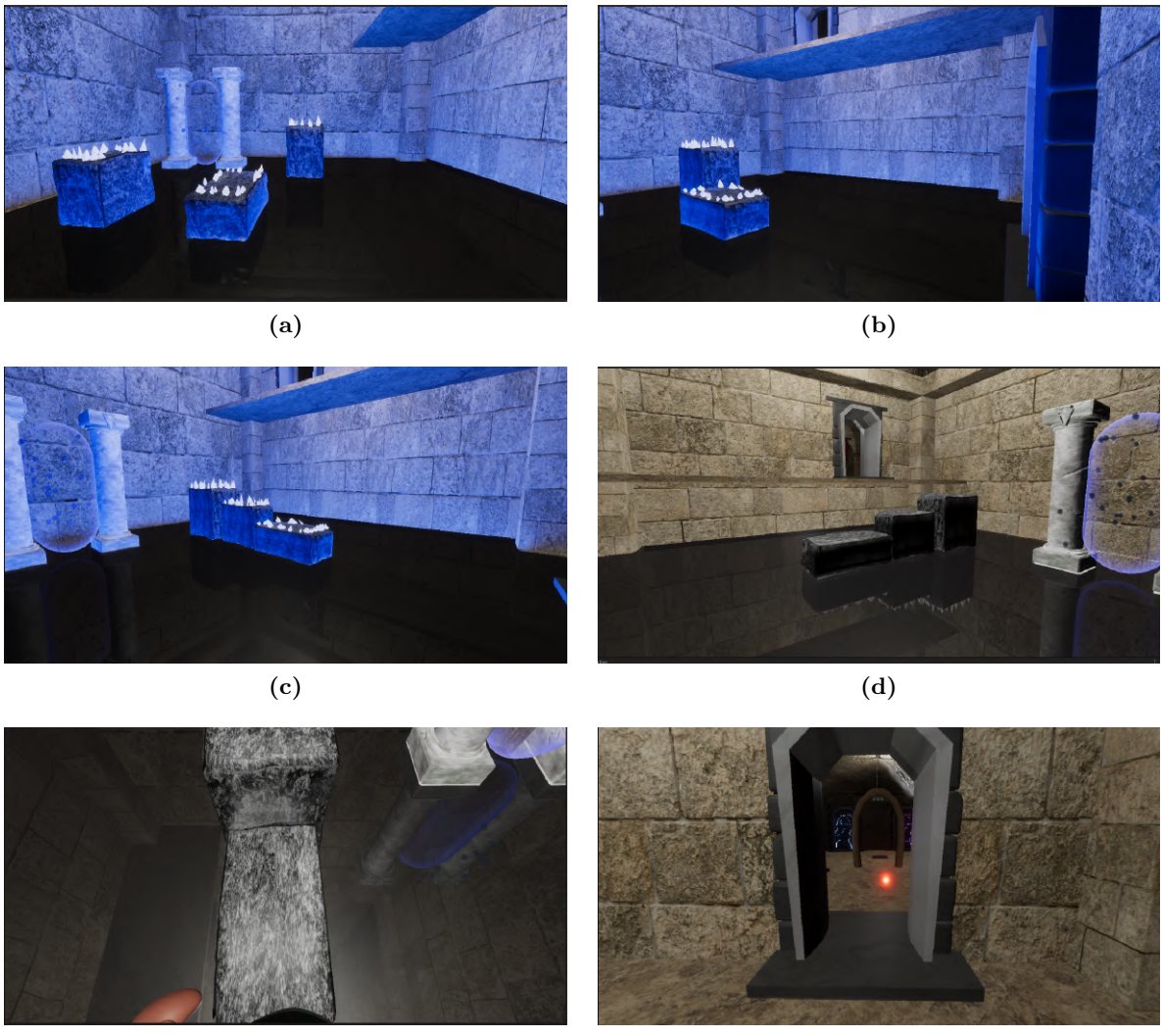
7.5 Puzzle example

As already mentioned, I came up with several combinations of game elements that were then used to design the individual puzzles. Using the modular components and implemented classes, I created the puzzles in the game. In this part, I show an example of a puzzle in the seventh room of the tutorial level. I describe the room and the proper walkthrough.

Room 0_7, the last room of the tutorial, finally makes the player think about the puzzle. All other previous rooms were mainly about teaching the player the game logic and introducing the elements.

In the room, there are three mirroring stones and one portal. The stone placed in the parallel world is controlled by the Rigidbody component and the **MirroringObjectInfo** class and has the **AudioController** script assigned. Apart from the Rigidbody component, the up world stone is controlled by the **GravityChange** class. The portal that changes worlds has the **RotatingWorld** script assigned.

Figure 7.2 shows the right pathing of the player to solve the puzzle. The player enters the room in the parallel world (a). S/he is told that things that seem similar in both worlds may actually vary, which means that they can be, for example, placed in different heights. When the player looks around, s/he can see that the door in the up world is much lower than the door in the parallel world (b). S/he can also see three stones of different heights. The player must move the stones to create stairs to be able to jump on the platform in the up world (c). The stones are moved by *OnControllerColliderHit* method of the **ColliderController** class, which calls the *AddForce* method of the Rigidbody component. The player then must enter the portal, which triggers *OnTriggerEnter* class of the **RotatingWorld** and changes worlds.Finally, s/he must jump on the stones (e) and leave the room (f).

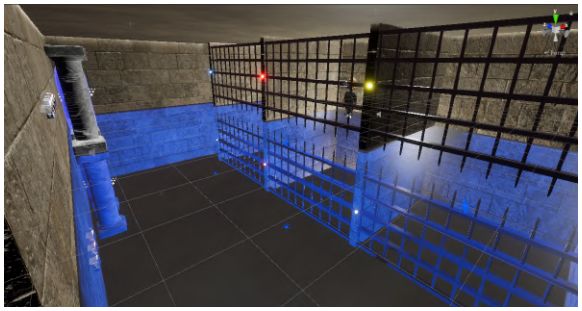


(a) **(b)**

(c) **(d)**

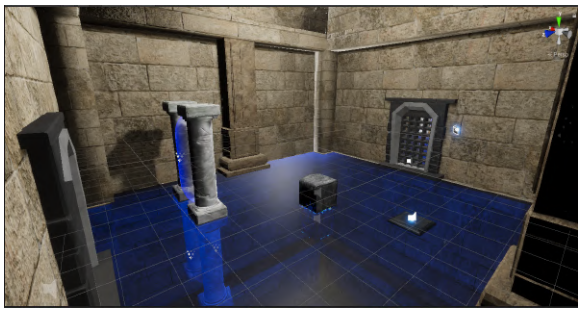
(e) **(f)**

Figure 7.2: Guide through room 0_7

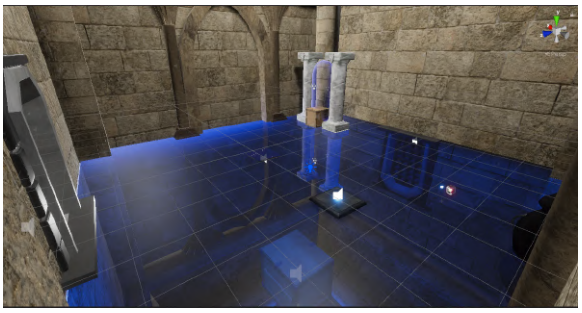


(a) The first room

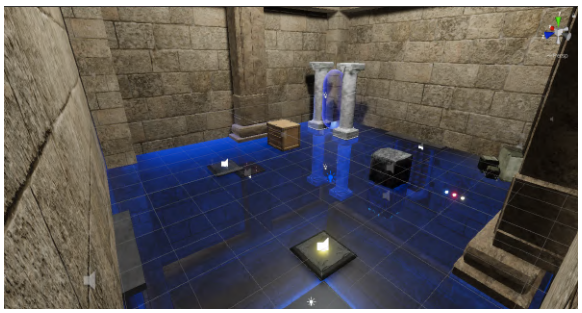
(b) Room 0_0



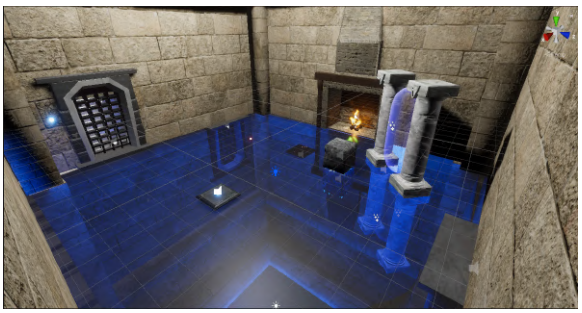
(c) Room 0_1



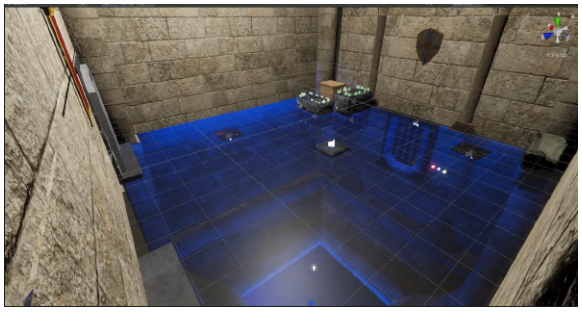
(d) Room 0_2



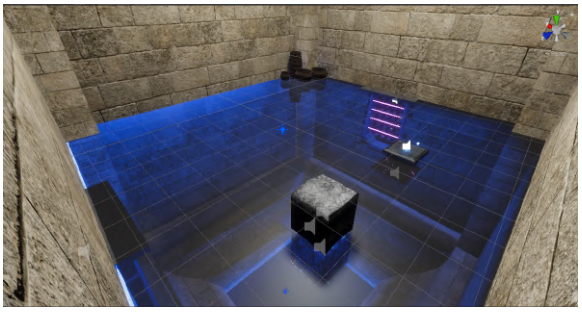
(e) Room 0_3



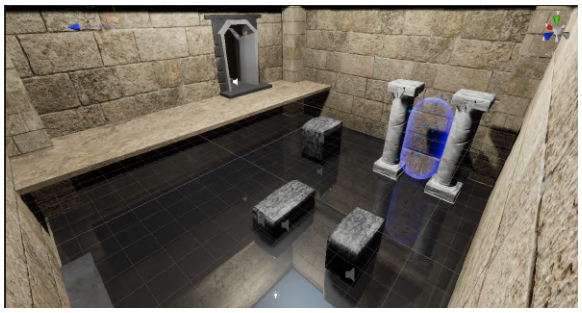
(f) Room 0_4



(g) Room 0_5



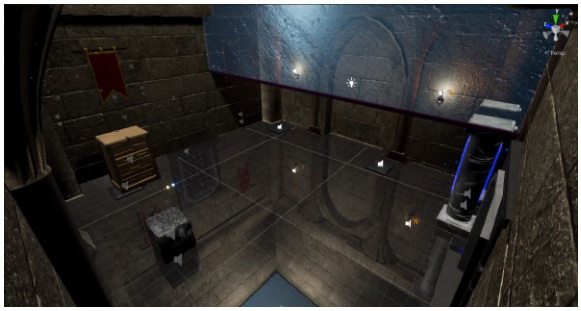
(h) Room 0_6



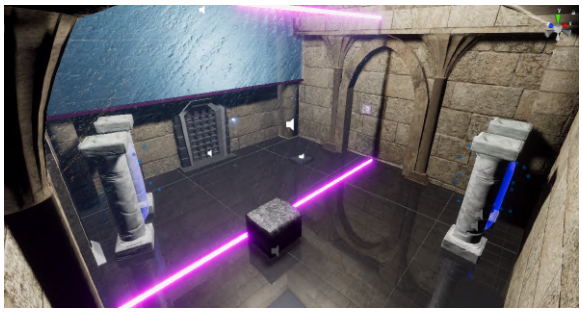
(i) Room 0_7

Figure 7.3: The tutorial level

(a) The Middle room



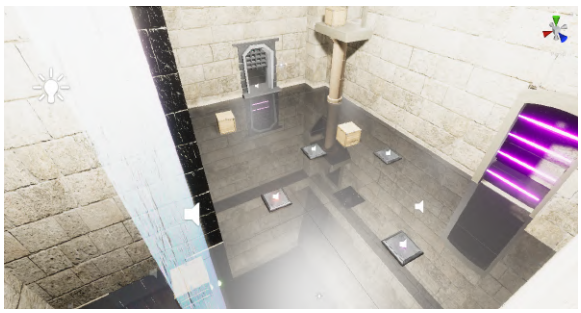
(b) Room 1_1



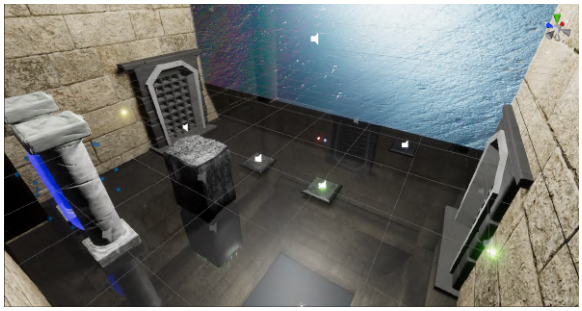
(c) Room 1_2

(d) Room 1_3

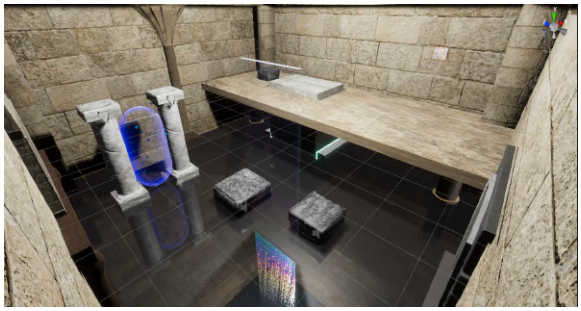
(e) Room 1_4



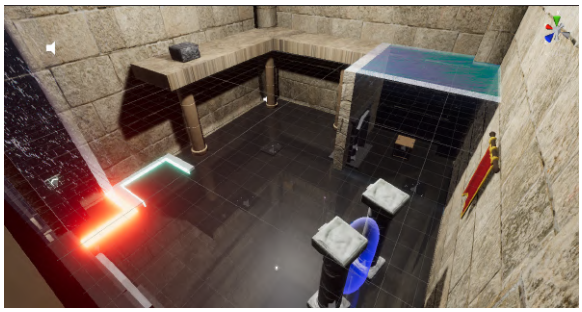
(f) Room 1_5



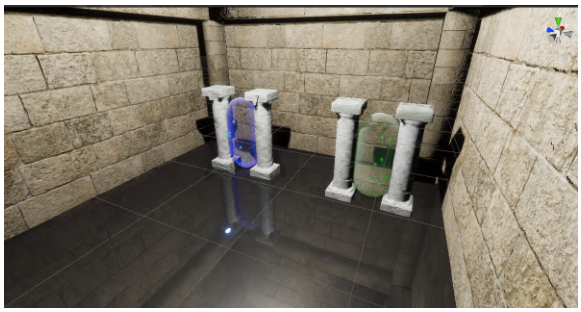
(g) Room 1_6



(h) Room 1_7



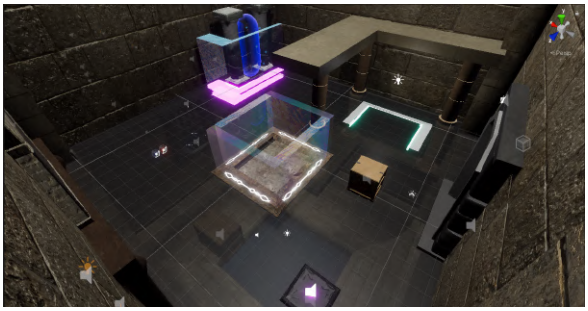
(i) Room 1_8



(j) Room 1_9

Figure 7.4: The first level

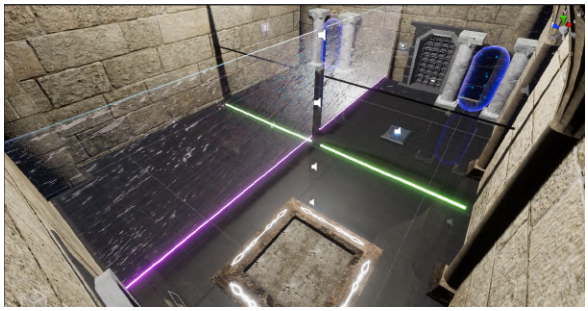
(a) Room 3_1



(b) Room 3_2

(c) Room 3_3

(d) Room 3_4



(e) Room 3_5

(f) Room 3_6

(g) Room 3_7

(h) Room 3_8

(i) Room 3_9

(j) Room 3_10

Figure 7.5: The second level

Chapter 8

Testing

In this chapter, I describe the testing methodology and the results of the tests. There were two tests—one during the game's development, particularly after creating the tutorial but before continuing with the subsequent levels. The other one was performed after almost finishing the development of the game.

8.1 Test 1

The purpose of the first test was to test the main theme of the game. To see if the puzzles are straightforward, if the explanation and hints are sufficient for the player to understand the rules but not too long to lose the player's attention. Another question is if the tutorial is not too easy or too difficult and whether the game preserves logic and it is not too chaotic. Another part of the testing was to check bugs and problems with the gameplay.

The game included only the tutorial level, containing 7 rooms with puzzles and one entry room. The game had no sound and graphically was only in an early stage. The tutorial monologue was partly implemented to test if the participants can understand the basics of the game only from the added text.

8.1.1 Test Scenario

This qualitative research was done with three participants, who had to go through all rooms of the tutorial level. The testing was performed on the participants' computers. Installing anything was not necessary. Due to the COVID-19 pandemic and related government regulations, the testing was done remotely. With my assistance, they downloaded a folder with the application in .exe format. The testers shared their screen using Skype or Discord, and the stream was being with their permission filmed using OBV studio.

The participants have a different level of experience with playing computer games, so the way of playing could differ.

The test started with a brief explanation of the game and what the player could expect. I also mentioned that the game was not finished, mainly the graphics like textures and models, and the participants should concentrate mainly on the gameplay.

To save time, the game has been modified before the testing in a way that the testers could continue with the game if a bug occurred. Doors dividing the rooms are the main element determining if the player can continue to the next room. Suppose the door stays closed even though it should open, then the tester is stuck and has to restart the game. A restoring system has not yet been implemented. This would immensely slow down the tester, and, therefore, the doors' colliders had been disabled. The game was also changed after the first participant tested it due to a missing feature. Without the feature, the game became, on a few occasions, impossible to finish. When the player pushed a crate to a wall, it was impossible to push that

crate away. It happened many times, and each time, the tester had to restart the game. In that case, I added pulling into the game. Moreover, a critical bug occurred that I thought would be better if fixed before continuing with testing. While changing the worlds, the worlds' animation made the player slightly shifted, which heavily impacted the gameplay.

8.1.2 Test Results

All participants were between the ages of 21 to 23. Two of the participants identify as women and one as a man. Their experience with computers and games was diverse. One of the participants considers his experience with computers and games as very high, and one of them considers her experience as occasional both with computers and computer games. The last participant answered as having an average experience with computers but having high experience playing video games.

Two respondents saw game logic as easily understood. The third respondent does not have problems with the game logic but thinks that to help the player to understand it, the worlds should be more different.

The participants had few reproaches about the visual side of the game. Two would appreciate it if the worlds would differ a bit more and the player could recognize the world at first glance. One of them says that objects should be visually more differentiated between the worlds to be clear in which world the player is. .She said: "The aesthetic side of the game lags behind the functionality." She would like the game better if the visual part was a bit more evolved. She thinks that the materials should differ in both worlds. She would also appreciate more game objects in the rooms to make them look better and not too empty.

No one had problems with controlling the game. It was clear to everyone how to move and how to perform basic tasks. Two players had problems keeping attention to the comments, and that led to not being sure what the next task was. One of them described the comments as "too much text". One tester also had problems due to bugs that were then fixed, which led to inconsistent behavior. And one participant did not determine how to pull a crate even though she knew which keyboard control to use and tried pressing it.

Two participants had problems understanding which crate can be moved in which world and sometimes tried to move the mirroring crate in the up world, even though it can be moved only in the parallel world. Two testers said that they had trouble recognizing in which world they currently were, and that led to getting lost in what crate they can currently move. Once they got to know the game rules and connected certain things, they did not have issues with following the gameplay and solving the puzzle.

Puzzles seemed to two testers as straightforward, they did not have problems distinguishing what objects mirror, and they did not mind the concept of two pressure plates being close to each other but not mirroring. The third participant responded with the puzzles being primarily straightforward.

One player had no problem recognizing what the purpose of each object in the game was. One player thinks that it could be better and one would like a few changes. She did not have to try each object to see what it does but had difficulty knowing the purpose at first glance.

The tutorial overall seemed to one participant as a little complicated. One of them said it was okay, and one said it was too lengthy and could be a little shorter.

All participants would play the game again, not just the tutorial but also other levels, because not much happened in the tutorial.

Two participants encountered a bug that made the open doors close after several world rotations. One of them could get through a wall due to a missing collider that was supposed to be there. Two participants had to go through the portals two times to start the rotation. As already said, during the first testing, the game view shifted after several changes of the worlds. It did not happen again because it was fixed, and then tested the corrected version. Additionally, the missing pulling feature was a large problem that had to be solved immediately.

8.1.3 Test Summary

Based on the testing, it looks like the game logic is easily understood. The players have no problem solving the puzzles and noticing which objects control what.

I concluded that the game is enjoyable, and I can continue creating more levels. The tutorial level does not need much change. It looked that the visual part of the game needed the most changes. More realistic-looking materials should be created, more types of walls should be added, and more objects that would make rooms better looking could be modeled.

The worlds could be differentiated by color to prevent confusion in which world the player is. Mirroring and non-mirroring crates should vary.

The tutorial dialogue should be more significant to make the player read the text. It could also consist of videos, which would show the player basic information.

As mentioned, most of the bugs were fixed during the testing. Pulling functionality was also added during the testing.

8.2 Test 2

This test was the final test after the game has been almost completed. The purpose of the test was mainly about finding bugs, testing if all puzzles are solvable, and finding if the game is generally playable and enjoyable.

8.2.1 Test Scenario

This test was performed in the same way as the first test.

There were six participants who all had different roles. Two of these participants also took part in the first testing, so they already had experience with the game. It would be ineffective for them to play the tutorial level again, so they played the first level while the rest tested the tutorial level. Two participants who played the tutorial level also tried two more rooms from the first level.

There were five participants between the ages of 22 to 25, and one participant was 53 years old. Three participants identify as women, while the rest identify as men. Five participants have significant experience with playing games, including puzzle games. Two of these have common computer knowledge, and three are more experienced in computers. The last participant has only a little experience with games and no experience in puzzle games. She also stated that her computer knowledge is minimal.

The testing was performed mainly on computers with the same or higher parameters as listed: OS: Windows 10 64-bit, Processor: Intel Core i-5-4460 3.2 GHz, Memory: 8 GB RAM, Graphics: NVIDIA GeForce GTX 960

8.2.2 Test Results

Only one participant tested on a laptop with lower parameters, which led to inconsistent unexpected behavior. The dialogue text was loaded slowly, the mouse, on the other hand, was too fast, and it was uneasy to control the game. During almost every second rotation of worlds, the player kept falling out of the world. Using the loading option, which restarted the game and transported the player to the last room she entered, the tester was able to finish the tutorial level, but it was not enjoyable gameplay. Other testers did not experience this behavior.

One participant had problems with movement. She knew the controls, but she could not combine mouse movement and keyboard controls to move the character smoothly. She thinks that is not the game's problem, but because she does not have enough experience. Others understood how to control the character and had no problems with the movement, although three participants stated that it would be nice to be able to move quicker.

Two participants would appreciate a special mouse cursor and not having the default white pointer.

Four participants complained about dragging objects. They did not like that the player was faster than the dragged object, which resulted in the player letting go of the object. Only one person noticed the text indicating pulling on their own. Others had to be notified. All participants forgot that they were crouching when they were crouching and were surprised why they were so slow.

I noticed every participant trying to move the mirroring stone in the up world. When asked about it, they said they knew that the stone could be moved only from the parallel world but did not think about it and tried it thoughtlessly.

The two participants, who tested the first level, stated that the game was a little too difficult. The other four testers did not mind the difficulty. One participant said: "It is not difficult to solve the puzzles, but it is difficult to control the game." S/he would appreciate a third-person view, which could help with the movement.

Two out of the four participants who tried the first room of the first level had problems understanding that the mirroring stone cannot move when there is a barrier in the up world while moving the stone in the parallel world. These two participants also did not know that movable objects could be moved by other objects.

The players who took part in the first test appreciated the visual improvement. One stated: "It's much nicer than it was."

Every tester who tried the tutorial level had problems recognizing the pressure plate in the other world in room 0_6. They did not see the plate, and when they did, they did not know, they should move the stone on it.

Overall, the testers had no problem understanding the game logic. The only problem that occurred was that some of the participants did not stop to think about the puzzle and plan the walkthrough and only tried out the things they saw.

Except for the participant who had problems with the game due to the high requirements, every player liked the game and said they would play it again. One participant said: "I liked that when I figured out something, I had a feeling of satisfaction." In room 1_7, one participant would like the glass barrier a bit closer to the platform.

One player did not know he could also jump on a stone in the parallel world. He thought that the crystals placed on top of the stone disabled the player to jump on the stone.

Bugs

Four players experienced a few bugs. Three players fell out of the room during the rotation of worlds. One player, as stated before, had this problem throughout the game, which I attribute to the insufficient parameters of the gaming platform. Two players experienced this bug only once during the gameplay. They loaded the game, and the bug did not occur again.

Three players were able to, after some trying, move the mirroring stone through a barrier blocking the other world.

One person encountered a bug on a pressure plate reacting to a crate. The plate got triggered, but when the player stepped off the plate (the crate stayed on the plate), the plate stayed triggered, but the door closed. It helped to move the crate off and on the plate again.

8.2.3 Test Summary

The test showed that the game is overall playable and enjoyable. However, the game controls, more specifically dragging objects, should be altered.

To prevent the confusion in room 0_6 and to show the player that objects can interact with each other and that the mirroring stone cannot move if its path is blocked in one world, I have to redesign room 0_6.

To solve the problem of participants forgetting they were crouching, the player could see a white text signalizing crouching, and crouch only while holding a proper key.

The test detected several bugs, but all testers were able to finish their level. However, the testing showed that it is necessary to state the minimum requirements to be able to play the game smoothly. The game should be optimized as part of future work, including not using the HDRP project but changing it to URP. HDRP is too complex for this game, while URP is sufficient and much more performance-friendly.

Chapter 9

Conclusion

This thesis aimed to create a 3D first-person logic game in Unity.

To complete this task, I researched the principles used in 3D logic games. According to these principles, I analyzed one logic game with similar features to my game. Then, I got acquainted with the level design process, which set the structure of my development. Based on this analysis, the game design document was written. This part followed the analysis of the use of modular components in the level design, their creation, and material creation in the physically based illumination model. According to this analysis, I identified the modular components of my game and built their polygonal models. With the use of created or downloaded textures, I created the materials for the models. Finally, I implemented the game in Unity and tested it with seven users. Two series of tests were performed: one during the game development process to test the game logic, and the other at the end of the development.

Even though the game has an ending, I plan to add more levels and new elements in the future. Because the game is divided into levels, it is not difficult to extend it.

Bibliography

[1] "3D Modeling Techniques in Games". In: (). [Online]. URL: https://3d-ace.com/press-room/articles/3d-modeling-techniques-games (visited on 04/28/2021).

[2] 80.lv, ed. *Defining Environment Language for Video Games*. [Online]. URL: https://80.lv/articles/defining-environment-language-for-video-games/ (visited on 04/28/2021).

[3] Inc Autodesk. *Subdivision Surface Modeling*. [Online]. 2010. URL: http://images.autodesk.com/adsk/files/subds.pdf (visited on 04/28/2021).

[4] "Basic Illumination Models". In: (May 2020). [Online]. URL: https://www.geeksforgeeks.org/basic-illumination-models/ (visited on 05/01/2021).

[5] *Blender*. [Online]. URL: https://www.blender.org/ (visited on 01/13/2021).

[6] *Blender 2.92 Reference Manual*. [Online]. 2021. URL: https://docs.blender.org/manual/en/latest/ (visited on 04/28/2021).

[7] Joel Burgess. *Skyrim's Modular Approach to Level Design*. [Online]. URL: https://www.gamasutra.com/blogs/JoelBurgess/20130501/191514/Skyrims_Modular_Approach_to_Level_Design.php (visited on 04/28/2021).

[8] J. Clement. *Number of games released on Steam worldwide from 2004 to 2020*. [Online]. Feb. 2021. URL: https://www.statista.com/statistics/552623/number-games-released-steam/ (visited on 05/09/2021).

[9] Shady Dave. *abstract (ambient loop)*. [Online]. URL: https://freesound.org/people/ShadyDave/sounds/345838/ (visited on 04/28/2021).

[10] *Defining Environment Language for Video Games*. [Online]. URL: https://www.unchartedthegame.com/en-us/media-gallery/ (visited on 04/28/2021).

[11] Zelda Dungeon, ed. *The Wind Waker Walkthrough – Dragon Roost Cavern*. [Online]. URL: https://www.zeldadungeon.net/the-wind-waker-walkthrough/dragon-roost-cavern/ (visited on 01/14/2021).

[12] Square Enix. *Lara Croft Go*. [Online]. URL: https://square-enix-games.com/ (visited on 01/13/2021).

[13] Claire Heginbotham. "What is 3D Digital Sculpting?" In: (). [Online]. URL: https://conceptartempire.com/what-is-3d-sculpting/ (visited on 04/28/2021).

[14] Hendryk Jaroslawsky. "Modular level design: A round up of the basics for budding level designers". In: (Feb. 2013). [Online]. URL: https://alumni.sae.edu/2013/02/08/modular-level-design-a-round-up-of-the-basics-for-budding-level-designers/ (visited on 04/28/2021).

[15] Raph Koster. *A Theory of Fun for Game Design*. Paraglyph Press, Inc, 2005. ISBN: 1-932111-97-2.

[16] Adam Kramarzewski and Ennio De Nucci. *Practical Game Design*. Packt Publishing, 2018. ISBN: 978-1-78712-179-9.

[17] Typographer Mediengestaltung. [Online]. 1999. URL: https://www.1001fonts.com/grange-font.html (visited on 04/28/2021).

[18] *Mixamo*. [Online]. URL: https://www.mixamo.com/#/ (visited on 04/28/2021).

[19] *Model without compromise*. [Online]. URL: https://solidedge.siemens.com/en/solutions/products/3d-design/subdivision-modeling/ (visited on 04/28/2021).

[20] "PBR: Theory". In: (). [Online]. URL: https://learnopengl.com/PBR/Theory (visited on 05/01/2021).

[21] Lee Perry. "Modular Level and Component Design Or: How I Learned to Stop Worrying and Love Making High-Detail Worlds". In: (Nov. 2002). [Online]. URL: https://docs.unrealengine.com/udk/Three/rsrc/Three/ModularLevelDesign/ModularLevelDesign.pdf (visited on 04/28/2021).

[22] Brian Pletcher. "Video Game History". In: *BRIAN'S DAMN PUZZLE BLOG* (Dec. 2009). [Online]. URL: http://mechanical-puzzles.blogspot.com/2009/12/nemesis-factor.html (visited on 01/13/2021).

[23] "Popular Modeling Methods in Filming and Gaming Industry". In: (). [Online]. URL: https://3d-ace.com/press-room/articles/modeling-methods-filming-and-gaming (visited on 04/28/2021).

[24] Gabriel Aguiar Prod. *Unity Shader Graph - Fire Flames Shader Tutorial*. [Online]. Dec. 2018. URL: https://www.youtube.com/watch?v=glSsaRpHKos&t=11s (visited on 04/28/2021).

[25] Scott Rogers. *Level Up! The Guide to Great Video Game Design*. John Wiley Sons, Ltd, 2010. ISBN: 978-0-470-68867-0.

[26] Rubik's, ed. *Faster Action No Sticker Cube launches*. [Online]. URL: https://www.rubiks.com/en-eu/about (visited on 01/14/2021).

[27] Jesse Schell. *The Art of Game Design*. Morgan Kaufmann, 2008. ISBN: 978-0-12-369496-6.

[28] Free Sound. [Online]. URL: https://freesound.org/ (visited on 04/28/2021).

[29] Unity Technologies, ed. *Unity*. [Online]. URL: https://unity.com/ (visited on 01/14/2021).

[30] *Texture Haven*. [Online]. URL: https://texturehaven.com/ (visited on 04/28/2021).

[31] Tribune, ed. *Sample of Jumble*. [Online]. URL: https://tribunecontentagency.com/article/sample-for-jumble/ (visited on 01/14/2021).

[32] *Unity Asset Store*. [Online]. URL: https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014#description (visited on 04/28/2021).

[33] *Unity User Manual*. [Online]. 2020. URL: https://docs.unity3d.com/Manual/ (visited on 01/13/2021).

[34] "Using Fresnel in your Materials". In: (). [Online]. URL: https://docs.unrealengine.com/en-US/RenderingAndGraphics/Materials/HowTo/Fresnel/index.html (visited on 05/01/2021).

[35] Jiří Žára et al. *Moderní počítačová grafika, 2. vydání*. Computer Press, 2004. ISBN: 80-251-0454-0.